Machine Learning from Human Preferences

Machine Learning from Human Preferences

Sang T. Truong, Andreas Haupt, and Sanmi Koyejo

Stanford University
Stanford, CA

Table of Contents

Introduction

Machine learning is increasingly shaping various aspects of our lives, from education and healthcare to scientific discovery. A key challenge in developing trustworthy intelligent systems is ensuring they align with human preferences. Learning from human feedback offers a promising solution to this challenge. This book introduces the foundations and practical applications of machine learning from human preferences. Instead of manually predefining the learning goal, the book presents preference-based learning that incorporates human feedback to guide the learning process, drawing insights from related fields such as economics, psychology, and human-computer interaction. Throughout, we emphasize not only the methods themselves but also their assumptions, limitations, and the conditions under which they can and cannot be applied—understanding when a model fails is as important as understanding when it works. By the end of this book, readers will be equipped with the key concepts and tools needed to design systems that effectively align with human preferences.

The book is intended for researchers, practitioners, and students who are interested in integrating machine learning with human-centered application. We assume some basic knowledge of probability and statistics, but provides sufficient background and references for the readers to follow the main ideas. The book also provides illustrative program examples and datasets. The field of machine learning from human preference is a vibrant area of research and practice with many open challenges and opportunities, and we hope that this book will inspire readers to further explore and advance this exciting field.

We hope with the present book to both allow more use of human preferences in machine learning, and new data modalities as Artificial Intelligence systems become increasingly important.

Stanford, May 2025, THK

Structure of this book

The book has three parts which introduce foundational models, present learning paradigms, and discuss societal considerations.

Part 1: Foundations

Chapter 1 lays the mathematical groundwork for the rest of the book. It covers random preference models, types of comparison data (binary rankings, accept-reject, lists), and deterministic

and stochastic utility models including the Rasch model, Bradley-Terry, and Gaussian Processes. A central theme is the Independence of Irrelevant Alternatives (IIA) axiom: how it simplifies modeling, its connection to modern methods like DPO and Elo, and its key limitations due to population heterogeneity.

Part 2: Learning

The second part introduces approaches to learning from and acting on comparisons.

- **Chapter 2** studies how to infer utility from preference data. It covers maximum likelihood estimation, Bayesian inference via MCMC and Laplace approximation, and online learning through Elo ratings as stochastic gradient descent. The chapter also addresses regularization, model selection via cross-validation, and modern optimization methods, with a case study on LLM preference learning.
- **Chapter 3** considers active data collection with the goal of efficient preference elicitation. Fisher information quantifies how much each comparison teaches us, enabling optimal query selection via A-optimal, D-optimal, and E-optimal design criteria. The chapter includes applications to robotic trajectory learning and active DPO for language model alignment.
- **Chapter 4** studies how learned preferences guide sequential decisions under uncertainty. It introduces Thompson Sampling for both linear and nonlinear objectives, dueling bandits with distinct winner concepts (Condorcet, Borda, von Neumann), preferential Bayesian optimization, and human-agent cooperation through CIRL.

Part 3: Society

The final part of the book addresses the societal dimensions of preference learning: how to aggregate diverse preferences and whose preferences should count.

- **Chapter 5** examines preference aggregation when multiple stakeholders disagree. It introduces social choice theory, Arrow's and Gibbard-Satterthwaite's impossibility theorems, and ways to escape these impossibilities through domain restrictions and scoring rules. It connects the Borda count to DPO, discusses nosy preferences and the liberal paradox, analyzes Community Notes as a case study, and covers mechanism design for incentive-compatible preference elicitation.
- **Chapter 6** asks a normative question: *whose preferences matter?* It shows how technical design choices at each stage of the preference learning pipeline embed value judgments. The chapter covers individual and group fairness, how unfairness compounds across pipeline stages, and design principles for responsible preference learning systems, including human values, AI alignment, and human-centered design.

Chapter 7 concludes the book by reflecting on its interdisciplinary approach and discussing open challenges, including moving beyond pairwise comparisons, scalable oversight, heterogeneity and personalization, fairness, and foundation model alignment.

How to engage with this book

There are four models of reading, and teaching with, this book. Chapter 1 is foundational to all of the book, so is part of all of these pathways.

- For practitioners and those teaching applied AI content, we recommend Chapters 1, 2, and 4, which can be used as a sequence in an early graduate course on Machine Learning. This sequence covers foundations, learning methods, and decision-making with human preferences.
- For people with background in discrete choice, we propose to skim Chapter 1, and study Chapters 2 and 4. These studies allow readers to integrate machine learning in their studies of discrete choice, demand models, and Industrial Organization.
- For those with deep background in machine learning, we propose to study Chapters 2-4. These chapters maximize the amount of machine learning covered, and are suitable for a deep learning-based course on machine learning.
- For those interested in the societal and theoretical foundations of machine learning from comparisons, we recommend Chapters 1, 5, and 6. Chapter 1 establishes the modeling foundations, Chapter 5 studies aggregation and mechanism design, and Chapter 6 examines fairness and value alignment. This pathway is suitable for a course on Computation and Society.

For instructors

This book is designed to support course instruction, and each chapter includes lecture plan callouts with suggested timing. A few notes for instructors adopting this book:

- **Pacing**: The book can be covered in a 10-week quarter (as in Stanford's CS329H) or a 15-week semester. For a quarter: allocate approximately 2 weeks each for Chapters 1–2, 1.5 weeks for Chapter 3, 2.5 weeks for Chapter 4, and 1 week each for Chapters 5–6. For a semester, expand each chapter proportionally.
- **Core vs. optional material**: The "Beyond Bradley-Terry" sections in Chapter 1 and the RL foundations section in Chapter 4 (marked "Optional") can be skipped without loss of continuity. All other material is core.
- **Assumptions and limitations**: Throughout the book, callout boxes marked with warnings highlight key assumptions and their limitations. We encourage instructors to treat these as first-class content—understanding when models fail is as pedagogically important as understanding when they work.

- **Cross-chapter connections**: Several results connect across chapters in non-obvious ways: Elo is SGD on the Bradley-Terry likelihood (Ch1–2); DPO implicitly optimizes the Borda score (Ch2–5); Fisher information drives both active learning (Ch3) and fairness auditing (Ch6). Emphasizing these connections reinforces the book's unified perspective.
- **Assessment**: Each chapter includes exercises at three difficulty levels, discussion questions for class participation, and quick check questions for self-assessment. Problem sets are available on the Teaching Materials[1] page.

Prior knowledge

The book assumes knowledge of the fundamentals of statistics, linear algebra and machine learning. Many example code excerpts are written in `python`, and make experience in the `python` programming language valuable for readers.

Recommended background texts

For readers seeking to strengthen their prerequisites or deepen their understanding of the fields this book draws from, we recommend the following companion texts:

- **Abu-Mostafa, Magdon-Ismail, and Lin**, *Learning from Data*[2] — An exceptionally clear introduction to the core ideas of machine learning, emphasizing generalization, bias-variance tradeoffs, and the limits of learning from finite data. Ideal preparation for the statistical learning concepts in Chapters 2–4.
- **Bishop**, *Pattern Recognition and Machine Learning*[3] — Comprehensive coverage of Gaussian processes, Bayesian inference, and logistic regression, all of which appear throughout this book.
- **Russell and Norvig**, *Artificial Intelligence: A Modern Approach*[4] — The standard AI textbook, providing context for the reinforcement learning and decision-making concepts in Chapter 4.
- **Sen**, *Collective Choice and Social Welfare*[5] — For readers wanting depth in social choice theory and welfare economics before Chapter 5, Sen's classic text provides the rigorous foundations.

[1] src/teaching.qmd
[2] https://amlbook.com/
[3] https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/
[4] https://aima.cs.berkeley.edu/
[5] https://www.cambridge.org/core/books/machine-learning/621D3E616DF879E494B094CC93ED36A4

Additional Materials

Every chapter has problems for readers and slides for teaching of the material available. See the Teaching Materials[6] page for all slides, problem sets, and suggested course pathways.

Acknowledgments

Initial versions of this book were compiled as lecture notes to the class CS329H: Machine Learning from Human Preferences at Stanford University taught in Fall 2023 and Fall 2024. We thank Rehaan Ahmad, Ahmed Ahmed, Jirayu Burapacheep, Michael Byun, Akash Chaurasia, Andrew Conkey, Tanvi Deshpande, Eric Han, Laya Iyer, Adarsh Jeewajee, Shreyas Kar, Arjun Karanam, Jared Moore, Aashiq Muhamed, Bidipta Sarkar, William Shabecoff, Stephan Sharkov, Max Sobol Mark, Kushal Thaman, Joe Vincent, Yibo Zhang, Duc Nguyen, Grace Sodunke, Ky Nguyen, and Mykkel Kochenderfer for their early contributions and feedback.

[6] src/teaching.qmd

1 Foundations

Intended Learning Outcomes

By the end of this chapter you will be able to:

- **Identify** where preference learning appears across machine learning: recommender systems, information retrieval, robotics, language model alignment, and ranking systems.
- **Distinguish** between *item-wise* preference data (accept/reject, ratings) and *pairwise* comparison data, understanding when each is appropriate.
- **Differentiate** deterministic preferences from *stochastic (random)* preferences and justify why randomness is essential for modeling noisy human choice.
- **Formulate** the *Rasch model* as the simplest factor model for item-wise data, identifying user appetite (U_i) and item appeal (V_j) as latent parameters.
- **Extend** 1D item utilities to *K-dimensional factor models*, explaining why multi-dimensional representations are essential for recommender systems.
- **Derive** how *Bradley-Terry emerges* as a special case of the Rasch model when conditioning on a single user.
- **Define** and **apply** the *Independence of Irrelevant Alternatives (IIA)* axiom, explaining how it collapses the full preference distribution to an M-parameter logit model.
- **Derive** choice probabilities for binary comparisons (Bradley–Terry), accept–reject decisions (logistic regression), and full or partial rankings (Plackett–Luce) from a random-utility model with i.i.d. Gumbel shocks.
- **Connect** the *Bradley–Terry model* to modern methods like *Direct Preference Optimization (DPO)* and *Elo ratings*.
- **Simulate** preference data from both item-wise (Rasch) and pairwise (Bradley-Terry) models, and **visualize** response matrices.
- **Diagnose** two key limitations of IIA—population heterogeneity (mixture models) and the *red-bus/blue-bus* cloning problem—and **motivate** richer models with correlated utility shocks.
- **Explain** why *Gaussian Processes (GPs)* provide a flexible nonparametric alternative to linear reward models, and identify when GPs are preferred over parametric approaches.

Video Overview

A visual tour of the key concepts in this chapter — from response matrices and the Rasch model to Bradley-Terry, IIA, and Gaussian process preference models.

../animations/ch1/chapter1_narrated.mp4

A consolidated table of mathematical notation used throughout this book is provided in the Mathematical Notation[1] appendix.

[1] notation.qmd

This is a book about machine learning from human preferences. Preference data arises throughout machine learning: recommender systems learn which products users prefer, search engines learn which results are most relevant, robots learn which trajectories humans find natural, and language models learn which responses are helpful. This chapter lays the theoretical foundation for understanding how preference data can be modeled and used for learning across all these domains.

1.1 Preference Learning Across Machine Learning

Preference data appears in many forms across machine learning applications:

- **Recommender systems**: Users implicitly express preferences through clicks, purchases, ratings, and engagement time. A user clicking on item A rather than item B reveals $A \succ B$. Netflix's recommendation engine, Amazon's product suggestions, and Spotify's playlist generation all learn from such preference signals.
- **Information retrieval**: Search engines learn from user clicks which documents are relevant to queries. If a user searches for "best restaurants nearby" and clicks on the third result, this suggests the third result was preferred over the first two—at least for that query.
- **Robotics and control**: When teaching robots through demonstration or feedback, humans express preferences over trajectories. A human might indicate that a robot arm's smooth motion is preferred over a jerky one, or that one grasping strategy is better than another.
- **Language model alignment**: Large language models are fine-tuned using human preferences over responses. Given a prompt, annotators compare candidate outputs and indicate which is more helpful, harmless, or honest.
- **Game playing and sports**: The Elo rating system, used in chess and many video games, is fundamentally a preference model—each game outcome is a pairwise comparison revealing which player was "preferred" (stronger) in that match.

Despite the diversity of applications, these all share a common mathematical structure: *pairwise comparisons* or *choices from sets* that reveal underlying preferences. The Bradley-Terry model and its extensions provide a unified framework for all these settings.

1.2 A Running Example: Language Model Alignment

To ground our discussion, we use language model alignment as a detailed running example. The ideas apply broadly, but LLMs provide concrete notation and have driven recent interest in preference learning.

Modern large language models like GPT-4 and Claude are trained in two distinct phases. Understanding this pipeline illustrates why preference models are essential to building AI systems that behave as humans intend.

Pretraining: Learning to Predict. In the first phase, called *pretraining*, a language model learns to predict the next token in a sequence. Given a corpus of text, the model is trained to minimize the cross-entropy loss:

$$\mathcal{L}_{\text{pretrain}} = -\mathbb{E}_{x \sim \mathcal{D}} \left[\sum_{t=1}^{T} \log \pi_\theta(x_t \mid x_{<t}) \right] \tag{1.1}$$

where $\pi_\theta(x_t \mid x_{<t})$ is the model's predicted probability of token x_t given the preceding tokens. This objective has a remarkable property: it is a *proper scoring rule*. A scoring rule is proper if a forecaster maximizes their expected score by reporting their true beliefs. For cross-entropy, this means that a model minimizing this loss will produce *calibrated* probability estimates—its predicted probabilities will match empirical frequencies in the training data.

Proper Scoring Rules

A scoring rule $S(p, y)$ assigns a score to a probability forecast p when outcome y is observed. The rule is *proper* if the expected score $\mathbb{E}_{y \sim q}[S(p, y)]$ is maximized when $p = q$, i.e., when the forecaster reports their true belief. The logarithmic scoring rule $S(p, y) = \log p(y)$ (negative cross-entropy) is strictly proper: any deviation from true probabilities strictly decreases expected score. This is why cross-entropy is the standard training objective for classification and language modeling—it incentivizes honest probability reporting.

Pretraining produces models that are remarkably capable at predicting text, but prediction alone does not ensure the model will behave helpfully, harmlessly, or honestly. A model trained only to predict text will happily continue any prompt, including harmful ones.

Posttraining: Aligning with Human Preferences. The second phase, called *posttraining* or *alignment*, teaches the model to produce outputs that humans prefer. This requires a fundamentally different kind of data: rather than predicting what text comes next, we need to learn *which outputs are better than others*.

The dominant approach for posttraining is *Reinforcement Learning from Human Feedback (RLHF)* Christiano et al. (2017) Ouyang et al. (2022). At a high level, RLHF works as follows:

1. **Collect preference data**: Given prompts x, sample pairs of responses (y_1, y_2) from the model and ask human annotators which response is better.
2. **Train a reward model**: Fit a reward function $r(x, y)$ to the preference data, predicting which response humans will prefer.
3. **Optimize the policy**: Fine-tune the language model to maximize the learned reward while staying close to the original model (to prevent reward hacking).

A recent and influential simplification is *Direct Preference Optimization (DPO)* Rafailov et al. (2023), which skips the explicit reward model. DPO directly optimizes the policy on prefer-

ence data using the objective:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right] \tag{1.2}$$

where y_w is the preferred ("winning") response, y_l is the dispreferred ("losing") response, π_{ref} is a reference policy (typically the pretrained model), and β controls how much the optimized policy can deviate from the reference.

The Bradley-Terry Connection. The key insight behind DPO is that it assumes human preferences follow a specific probabilistic model. If we define an implicit reward $r^*(x,y) = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$, then the DPO objective is equivalent to maximum likelihood estimation under the assumption:

$$p(y_w \succ y_l \mid x) = \sigma(r^*(x,y_w) - r^*(x,y_l)) = \frac{1}{1+e^{-(r^*(x,y_w)-r^*(x,y_l))}} \tag{1.3}$$

This is precisely the *Bradley-Terry model* Bradley and Terry (1952), one of the oldest and most widely-used models for pairwise comparisons. The Bradley-Terry model says that the probability of preferring item j over item j' is a sigmoid function of the difference in their "strengths" or utilities.

This raises natural questions: *Why is the Bradley-Terry model a reasonable assumption? Where does it come from? When does it fail?* Answering these questions is the goal of the rest of this chapter. We will see that Bradley-Terry arises naturally from a *random utility model* with specific noise assumptions, and that its key property—the *Independence of Irrelevant Alternatives*—is both powerful and limiting.

WORKED EXAMPLE: COMPUTING BRADLEY-TERRY PROBABILITIES

Suppose we have three chess players with Elo-style utilities $V_A = 2.0$, $V_B = 1.0$, and $V_C = 0.0$. Under the Bradley-Terry model:

Step 1. Compute pairwise win probabilities:

$$p(A \succ B) = \sigma(V_A - V_B) = \sigma(1.0) = \frac{1}{1+e^{-1}} \approx 0.731$$

$$p(A \succ C) = \sigma(V_A - V_C) = \sigma(2.0) = \frac{1}{1+e^{-2}} \approx 0.881$$

$$p(B \succ C) = \sigma(V_B - V_C) = \sigma(1.0) \approx 0.731$$

Step 2. Check IIA: Adding or removing player C does not change the probability that A beats B. This is a direct consequence of the model — $p(A \succ B)$ depends only on $V_A - V_B$.

Step 3. Compute Plackett-Luce ranking probabilities for a full ranking $A \succ B \succ C$:

$$p(A \succ B \succ C) = \frac{e^{V_A}}{e^{V_A} + e^{V_B} + e^{V_C}} \cdot \frac{e^{V_B}}{e^{V_B} + e^{V_C}} = \frac{e^2}{e^2 + e^1 + e^0} \cdot \frac{e^1}{e^1 + e^0} \approx 0.665 \times 0.731 \approx 0.486$$

Interpretation: The strongest player (A) is expected to win about 73% of games against the middle player (B) and 88% against the weakest (C). The most likely full ranking has A first, occurring about 49% of the time.

KEY TAKEAWAY: THE BRADLEY-TERRY MODEL

Whether aligning language models, building recommender systems, or ranking chess players, many preference learning methods assume the Bradley-Terry model: $p(j \succ k) = \sigma(V_j - V_k)$. Understanding *why* this model is reasonable—and when it fails—requires the theory of random utility models and the Independence of Irrelevant Alternatives. The same mathematical framework applies across all these applications.

SUGGESTED LECTURE PLAN

This chapter can be covered in two 50-minute lectures plus a discussion section:

Lecture 1 (Sections 1–3): Motivation and foundations

- Preference learning across ML: recommenders, search, robotics, LLMs (10 min)
- LLM alignment example: RLHF, DPO, and Bradley-Terry (20 min)
- Random preferences, types of comparison data (20 min)

Lecture 2 (Sections 4–6): Theory and limitations

- Random utility models, the Ackley function examples (20 min)
- IIA, Gumbel equivalence, derivation of choice probabilities (20 min)
- Limitations: heterogeneity and red-bus/blue-bus (10 min)

Discussion section: Exercises 1–3, code walkthroughs for Ackley simulations

1.3 Chapter Overview

Having motivated preference models through their role across machine learning—from recommender systems to language model alignment—we now develop the theoretical foundations. The rest of this chapter covers:

- **Random preferences** (): Why we model preferences as stochastic, and the mathematical framework for doing so.
- **Types of comparison data**: Full rankings, choices from sets, and binary comparisons—all as observations from the same underlying preference distribution.
- **Random utility models**: Representing preferences through utility functions with noise, and deriving choice probabilities.

- **Independence of Irrelevant Alternatives (IIA)**: The key assumption that makes Bradley-Terry and related models tractable, plus its limitations.

Chapters 3-6 will then cover how to *learn* preference models from data, *actively collect* informative comparisons, make *decisions* based on learned preferences, and handle *heterogeneous* populations with diverse preferences.

Historical Notes. The ideas in this chapter draw from nearly a century of research across psychology, economics, and statistics:

- **Thurstone (1927)** introduced the *law of comparative judgment* in psychophysics, proposing that perceived stimuli have random "discriminal processes"—the first random utility model.
- **Luce (1959)** provided an axiomatic foundation through his *choice axiom*, which is equivalent to IIA. This work established the mathematical foundations for probabilistic choice theory.
- **Bradley & Terry (1952)** developed the paired comparison model that bears their name, originally for ranking chess players and other competitive settings.
- **McFadden (1974)** connected random utility theory to econometrics, developing the conditional logit model for analyzing discrete choices. This work, along with his development of the nested logit and mixed logit models, earned him the Nobel Prize in Economics in 2000.
- **Modern ML**: The application of preference models to machine learning accelerated with Christiano et al. (2017)'s introduction of RLHF for training AI systems. Rafailov et al. (2023)'s DPO provided a direct connection between preference optimization and the classical Bradley-Terry model.

> "The theory of discrete choice…has become a cornerstone of modern microeconometrics, with applications ranging from transportation to marketing to environmental economics." — Daniel McFadden, Nobel Lecture (2000)

1.4 Random Preferences as a Model of Comparisons

We start with a set of **items** $j \in \{1, \dots, M\}$—be they products, robot trajectories, or language model responses. We will consider models to generate comparisons that are orders. For realism, but also for mathematical simplicity, we will assume in this book that the set of items is discrete and has M items.

Comparisons may be random and are generated by random draws of (total) orders. (Total) Orders have two properties.

- First, for two items j, j' either $j \prec j'$ and/or $j \succ j'$ must hold, an assumption called *totality*:[2] Either j is weakly preferred to j' or j' is weakly preferred to j.
- The second assumption is transitivity: if $j \succ j'$ and $j' \succ k$, then also $j \succ k$.

In the following, we consider randomness as generated from a *decision-maker* who has an order, or preference relation, $\prec$ on a set of items. We refer to the random object $\prec$ as the oracle preference. Each preference $\prec$ has an associated probability mass $p(\prec)$, leading to an $(M! - 1)$-dimensional vector encoding the full random preference distribution. This grows extremely fast: even with just $M = 4$ items, we need $4! - 1 = 23$ parameters; with $M = 10$, we need over 3.6 million. Reducing this complexity is a central goal of this chapter—we will see that the Independence of Irrelevant Alternatives collapses this to just M parameters.

One might wonder why we need to have a random preference. Deterministic preferences are conceptually helpful constructs and are used broadly in the fields of Consumer theory (e.g., Mas-Colell et al. (1995)). However, they suffer when bringing them to data, as data is inherently noisy.

Even when allowing for randomness, assumptions we will impose in this chapter—transitivity and the Independence of Irrelevant Alternatives—are strong yet practical. In many situations they will fail, for good reasons: humans cannot always clearly rank alternatives, their choices may reflect cultural norms, or they might exhibit self-control problems. These limitations of classical preference models will be discussed in later chapters. Until then, we will make full use of learning stochastic preferences.

Assumption: Totality and Its Limits

Totality assumes that for every pair of items, the decision-maker can express a preference (or indifference). In practice, humans often have **genuinely incomplete preferences**: they may lack the expertise to compare two items, the items may differ along incommensurable dimensions (e.g., helpfulness vs. safety in LLM responses), or the cognitive load of comparison may be too high. Forcing a total order on such preferences introduces noise that is not well-captured by random utility models—it is not randomness in preference, but absence of preference. Models that allow for incomparability (partial orders) exist but require substantially different estimation approaches; see Chapter 7 for open challenges.

1.5 Types of Comparison Data

There are different types of comparison data we may observe. We can relate them back to the population preferences $\prec$.

Full Preference Lists. The conceptually simplest and practically most verbose preference sampling is to get the full preference ranking, i.e., $L = (j_1, j_2, \dots, j_M)$, where $j_1 \succ j_2 \succ \cdots \succ j_M$.

[2] One often also allows for preferences to capture a notion of equivalence, called indifference, which is unlikely to happen in random choice models we will learn from data. We use the benefit of simplified notation and restrict to no indifference.

In this case, we know not only that j_1 is preferred to j_2, but also, by transitivity, that it is preferred to all other options. Similarly, we know that j_2 is preferred to all options but j_1, *etc.* In many cases, we do not observe full preferences as the cognitive load for humans is too high.

Choices from Subsets. Another type of sample is $(j, \mathcal{S})$ where j is the most preferred alternative from subset $\mathcal{S}$ for a sampled preference. Formally, $j \succ k$ for all $k \in \mathcal{S} \setminus \{j\}$—$j$ is preferred to all elements of $\mathcal{S}$ other than itself.

Formally, the probability that we observe $(j, \mathcal{S})$ is

$$p(j \mid \mathcal{S}) = \sum_{\prec : j \succ k \ \forall k \in \mathcal{S} \setminus \{j\}} p(\prec). \tag{1.4}$$

That is, the probability of observing $(j, \mathcal{S})$ is given by the sum of all preferred samples $\prec$ such that j is preferred to all k in $\mathcal{S}$ other than j.

If the choice is binary, $\mathcal{S} = \{j, j'\}$, we write this as $Y_{jj'} = 1$ (item j preferred over j'). We highlight that these outcomes are random, and depend on the sample of $\prec$. Binary data is convenient and quick to elicit and has been prominently applied in language model finetuning and evaluation.

Sometimes, particularly when a decision-maker is offered an item or "nothing", we will implicitly assume that there is an "outside option" indexed by 0, allowing us to interpret $Y_{j0} = 1$ as "accepting" item j, and $Y_{j0} = 0$ as rejecting it. Outside options can be thought of as fundamental limits to what a system designer can obtain. Consider a recommendation system. A user of that system might engage with content or not. In principle, instead of engaging, they will do something else. We do not model this explicitly as a fundamental abstraction. *All models are wrong, but some are useful.*

Item-wise Responses. In many applications, users respond to individual items rather than comparing pairs. We denote $Y_{ij} \in \{0, 1\}$ as the binary response of user i to item j, where $Y_{ij} = 1$ indicates acceptance (like, purchase, engagement) and $Y_{ij} = 0$ indicates rejection. This yields an $N \times M$ *response matrix*:

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1M} \\ Y_{21} & Y_{22} & \cdots & Y_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N1} & Y_{N2} & \cdots & Y_{NM} \end{bmatrix} \tag{1.5}$$

Examples of item-wise data include: e-commerce purchases (did user i buy product j?), streaming behavior (did user i play or skip track j?), dating apps (did user i swipe right on profile j?), and content moderation (did annotator i flag content j?). Item-wise data is often more abundant than pairwise data—every user-item interaction generates one data point—but requires modeling both user and item parameters to capture heterogeneity across users.

Context. Choices are often conditional, and data is given by (i, L) (for list-based data), $(i, j, \mathcal{S})$ (for general choice-based data), (i, j, j') for pairwise data, or (i, j) for item-wise data. Here i indexes the *user* or *context*: the environment of a purchase, the goal of a robot, or a user prompt for a large language model. It can also be a prompt to the decision-maker, e.g., to human raters

on whether they should pick preferences based on helpfulness or harmlessness Ganguli et al. (2022). The inclusion of context in learning allows for the generalization of preferences, as we will see in subsequent chapters.

Example: Preference Data Across Domains

Recommender Systems. A user browsing an e-commerce site sees products $\{A, B, C\}$ and clicks on B. This reveals the choice $(B, \{A, B, C\})$—item B was preferred from the displayed set. If we only track whether the user purchased, we observe accept-reject data: $Y_{B0} = 1$ (accepted B over the outside option of not buying).

Information Retrieval. A user searches "best pizza near me" and clicks on the third result. This provides implicit feedback that result 3 was preferred over results 1 and 2 for this query context.

Language Model Alignment. Given a prompt x = "Explain quantum entanglement to a 10-year-old," annotators compare two responses and indicate $A \succ B$. This triple $(x, y_w = A, y_l = B)$ constitutes one training example for RLHF or DPO. Public datasets in this format include Anthropic's HH-RLHF, OpenAI's summarization preferences, and the Stanford Human Preferences (SHP) dataset.

Sports Rankings. Each chess match between players j and k yields a pairwise comparison. If player j wins, we observe $Y_{jk} = 1$. The Elo rating system models these outcomes using the Bradley-Terry model.

In all cases, the mathematical structure is the same: observations reveal preferences over items, and our goal is to learn the underlying utility function.

1.6 Deterministic Utility Models

The preference models discussed so far treat items as having fixed utilities V_j. But in many applications—especially recommender systems—different users have different preferences over the same items. A horror movie fan and a romantic comedy fan will have very different utility for the same film. This section introduces **deterministic utility models** (also called latent variable models) that accommodate both user-specific and item-specific parameters.

From Items to Users and Items. When we observe binary responses $Y_{ij} \in \{0, 1\}$ from N users to M items, we model the response probability using a latent utility that depends on both user and item:

$$p(Y_{ij} = 1) = \sigma(H_{ij}), \quad H_{ij} = f(U_i, V_j) \tag{1.6}$$

where U_i captures user i's characteristics and V_j captures item j's characteristics. In this **deterministic utility framework**, stochasticity enters through the Bernoulli sampling $Y_{ij} \sim \text{Bernoulli}(\sigma(H_{ij}))$, not through noise in H_{ij} itself. The function f and the dimensionality of U_i, V_j define different model families.

1.6.1 The Rasch Model

The **Rasch model** (also called the 1-parameter logistic model in psychometrics) is the simplest factor model, where f is additive:

$$p(Y_{ij} = 1 \mid U_i, V_j) = \sigma(U_i + V_j) \tag{1.7}$$

Here:

- $U_i \in \mathbb{R}$ is the **user appetite**: user i's general tendency to accept items. High U_i means an enthusiastic user who likes many things; low U_i means a selective user.
- $V_j \in \mathbb{R}$ is the **item appeal**: how universally appealing item j is. High V_j means a crowd-pleaser; low V_j means a niche item.

The probability of acceptance depends only on the sum $U_i + V_j$: enthusiastic users accept more items, and popular items are accepted by more users.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| autorun: true
rng = np.random.default_rng(2601)

N, M = 50, 30  # 50 users, 30 items

# Sample latent parameters
U = rng.normal(loc=0.0, scale=1.0, size=N)  # user appetites
V = rng.normal(loc=0.0, scale=1.0, size=M)  # item appeals

# Compute response probabilities: p(Y_ij = 1) = sigmoid(U_i + V_j)
P = 1.0 / (1.0 + np.exp(-(U[:, None] + V[None, :])))

# Sample binary responses
Y = (rng.random(size=(N, M)) < P).astype(int)

# Sort by row/column means for visualization
row_order = np.argsort(Y.mean(axis=1))
col_order = np.argsort(Y.mean(axis=0))
Y_sorted = Y[row_order][:, col_order]
```

```
plt.figure()
plt.imshow(Y_sorted, cmap='coolwarm', aspect='auto')
plt.colorbar(label='Response (0=reject, 1=accept)')
plt.xlabel("Items (sorted by popularity)")
plt.ylabel("Users (sorted by appetite)")
plt.title("Rasch Model: Response Matrix")
plt.show()
```

The sorted response matrix reveals the structure: high-appetite users (top) accept most items, popular items (right) are accepted by most users, and the transition from rejection to acceptance follows a diagonal pattern determined by the sum $U_i + V_j$.

1.6.2 Connecting Rasch to Bradley-Terry

A remarkable fact connects item-wise and pairwise models: for a **single user**, the Rasch model implies Bradley-Terry for pairwise comparisons.

Derivation: Suppose user i makes a pairwise comparison between items j and k. In the Rasch framework, the user has deterministic utilities $H_{ij} = U_i + V_j$ and $H_{ik} = U_i + V_k$. When comparing, the user's binary responses $Y_{ij}, Y_{ik} \sim \text{Bernoulli}(\sigma(H_{ij})), \text{Bernoulli}(\sigma(H_{ik}))$ are noisy observations of these utilities.

One natural comparison mechanism is to prefer j over k if the user would accept j in isolation more readily than k. Under the **logistic difference model**, the pairwise preference arises from comparing the latent utilities with logistic noise:

$$p(j \succ k \mid i) = p(H_{ij} + \eta_{ij} > H_{ik} + \eta_{ik}) \tag{1.8}$$

where η_{ij}, η_{ik} are independent logistic noise terms (corresponding to the inverse CDF of the Bernoulli observation process). Since the difference of two logistic random variables is also logistic:

$$p(j \succ k \mid i) = \sigma(H_{ij} - H_{ik}) = \sigma((U_i + V_j) - (U_i + V_k)) = \sigma(V_j - V_k) \tag{1.9}$$

The user-specific parameter U_i **cancels out**! This has important implications:

! KEY INSIGHT: WHAT DIFFERENT DATA TYPES REVEAL

Data Type	What It Reveals	Parameters Identified
Pairwise ($j \succ k$)	Item differences only	$V_j - V_k$ (up to constant)
Item-wise (Y_{ij})	User appetites + item appeals	U_i and V_j (up to constant)

Pairwise data cannot distinguish between a world where all items are excellent and users are selective, versus all items are mediocre and users are enthusiastic. Item-wise data can make this distinction.

Proof: Rasch Implies Bradley-Terry

Goal: Show rigorously that the deterministic utility Rasch model with Bernoulli observations yields Bradley-Terry pairwise probabilities.

Setup: User i has deterministic utilities $H_{ij} = U_i + V_j$ for each item. Binary responses are:

$$Y_{ij} \sim \text{Bernoulli}(p_{ij}), \quad p_{ij} = \sigma(H_{ij}) = \frac{1}{1 + e^{-(U_i + V_j)}} \tag{1.10}$$

Pairwise comparison mechanism: When comparing items j and k, one natural model is the **logistic noise model**: we sample independent logistic noise η_{ij}, η_{ik} and prefer j if:

$$H_{ij} + \eta_{ij} > H_{ik} + \eta_{ik} \tag{1.11}$$

This corresponds to the observation that each Bernoulli response Y_{ij} can be viewed as thresholding a latent continuous variable with logistic noise:

$$Y_{ij} = \mathbb{1}[H_{ij} + \eta_{ij} > 0] \tag{1.12}$$

Derivation:

$$\begin{aligned} p(j \succ k \mid i) &= p(H_{ij} + \eta_{ij} > H_{ik} + \eta_{ik}) && (1.13)\\ &= p(\eta_{ij} - \eta_{ik} > H_{ik} - H_{ij}) && (1.14)\\ &= p(\eta_{ij} - \eta_{ik} > (U_i + V_k) - (U_i + V_j)) && (1.15)\\ &= p(\eta_{ij} - \eta_{ik} > V_k - V_j) && (1.16) \end{aligned}$$

Since η_{ij} and η_{ik} are independent logistic random variables, their difference $\eta_{ij} - \eta_{ik}$ is also logistic. Therefore:

$$p(j \succ k \mid i) = \sigma(V_j - V_k) \tag{1.17}$$

This is exactly the Bradley-Terry formula, and notice that U_i cancels out—pairwise probabilities depend only on item parameters V_j, V_k.

Interpretation: The Bernoulli observation model with deterministic utilities is **equivalent** to having noisy utilities with logistic noise for the purpose of pairwise comparisons.

This explains why recommender systems, which need to personalize, rely heavily on item-wise data (clicks, purchases, ratings), while ranking systems that only need to order items (chess ratings, LLM evaluation) can use pairwise comparisons.

1.6.3 K-Dimensional Factor Models

The Rasch model assumes users and items can each be characterized by a single number. This is limiting: a user might love action movies but dislike romance, while another has the opposite preference. To capture such heterogeneity, we extend to **K-dimensional** representations.

The Logistic Factor Model:

$$H_{ij} = U_i^\top V_j + Z_j \tag{1.18}$$

where:

- $U_i \in \mathbb{R}^K$ is the **user embedding**: user i's preferences along K latent dimensions
- $V_j \in \mathbb{R}^K$ is the **item embedding**: item j's characteristics along the same K dimensions
- $Z_j \in \mathbb{R}$ is the **item offset**: baseline popularity independent of user preferences

The dot product $U_i^\top V_j$ measures alignment between user preferences and item characteristics. When $K = 1$ and $Z_j = 0$, this reduces to a reparameterized Rasch model.

The Ideal Point Model:

$$H_{ij} = -\|U_i - V_j\|_2 + Z_j \tag{1.19}$$

Here, users and items live in the same K-dimensional space, and users prefer items *close* to their ideal point U_i. This model is natural for:

- Political preferences: voters (users) and candidates (items) on an ideological spectrum
- Music taste: listeners prefer songs similar to their preferred style
- Product recommendations: users prefer products near their ideal specifications

```
#| autorun: true
K = 3  # latent dimensions
rng = np.random.default_rng(42)

# User and item embeddings
U_k = rng.normal(0, 1.0, size=(N, K))  # N users × K dimensions
V_k = rng.normal(0, 1.0, size=(M, K))  # M items × K dimensions
Z = rng.normal(0, 0.5, size=M)          # item offsets

# Latent utilities via dot product
H = U_k @ V_k.T + Z[None, :]  # shape: (N, M)
P_factor = 1.0 / (1.0 + np.exp(-H))

# For pairwise data: compute all pairs for each user
# H_diff[i, j, k] = H[i, j] - H[i, k] = utility difference for user i comparing j vs k
H_diff = H[:, :, None] - H[:, None, :]  # shape: (N, M, M)
P_pair = 1.0 / (1.0 + np.exp(-H_diff))

print(f"Item-wise data shape: {P_factor.shape}  (N × M = {N} × {M})")
print(f"Pairwise data shape:  {P_pair.shape}  (N × M × M = {N} × {M} × {M})")
print(f"\nItem-wise: {N * M:,} possible observations")
```

```
print(f"Pairwise:  {N * M * (M-1) // 2:,} possible observations per user")
```

Notice that pairwise data grows as $O(M^2)$ per user, while item-wise data grows as $O(M)$. This combinatorial explosion makes exhaustive pairwise comparison impractical for large item sets.

1.6.4 Why Factor Models Matter

Factor models are the foundation of modern recommender systems:

1. **Matrix Factorization** (Netflix Prize): The winning approaches learned low-rank factorizations $Y \approx UV^\top$ of the user-item rating matrix.
2. **Collaborative Filtering**: Similar users have similar embeddings $U_i \approx U_{i'}$; similar items have similar embeddings $V_j \approx V_{j'}$. This enables predicting preferences for unobserved user-item pairs.
3. **Two-Tower Models**: Modern industrial systems use neural networks to compute $U_i = f_\theta(\text{user features})$ and $V_j = g_\phi(\text{item features})$, then score via dot product.
4. **Low-Rank Structure**: The $N \times M$ response matrix can be approximately reconstructed from $N \times K$ user embeddings and $M \times K$ item embeddings, where typically $K \ll \min(N, M)$. This compression enables generalization.

We defer *learning* factor model parameters to Chapter 3, but understanding their structure is essential for modeling preference data.

⚠ LIMITATIONS OF LINEAR FACTOR MODELS

The factor model $H_{ij} = U_i^\top V_j + Z_j$ assumes utility is a **linear function** of latent embeddings. This means it cannot capture interactions between feature dimensions—for instance, a user who values both "conciseness" and "formality" independently, but dislikes responses that are both concise *and* formal. More generally, dot-product models impose a monotonic relationship along each latent dimension. Nonlinear alternatives (such as the ideal point model $H_{ij} = -\|U_i - V_j\|_2 + Z_j$ or neural network scoring functions) can capture richer preference structures, but at the cost of identifiability and computational complexity. The Gaussian Process models introduced later in this chapter provide a flexible nonparametric alternative.

! KEY TAKEAWAY: THE LATENT VARIABLE VIEW

The latent variable view extends preference models to handle user heterogeneity:

- **Rasch model**: $p(Y_{ij} = 1) = \sigma(U_i + V_j)$ — 1D user appetite + item appeal
- **Factor model**: $H_{ij} = U_i^\top V_j + Z_j$ — K-dimensional embeddings
- **Ideal point**: $H_{ij} = -\|U_i - V_j\|_2 + Z_j$ — proximity in latent space

For a single user, Rasch implies Bradley-Terry for pairwise comparisons. But item-wise data reveals richer

structure (both U_i and V_j) than pairwise data (only $V_j - V_k$).

1.6.5 Two Equivalent Views of Stochastic Choice

Before introducing random utility models formally, we clarify an important conceptual point. The **latent variable models** above and the **random utility models** we introduce next represent **two equivalent ways to model the same stochastic choice behavior.**

1.6.5.1 Deterministic Utility View (Latent Variable Framework)

In latent variable models (such as the Rasch and factor models), utilities are **deterministic**:

$$H_{ij} = f(U_i, V_j) \quad \text{(no randomness)} \tag{1.20}$$

Stochasticity enters through the **observation process**:

$$Y_{ij} \sim \text{Bernoulli}(\sigma(H_{ij})) \tag{1.21}$$

For pairwise comparisons, when f is additive (e.g., $f(U_i, V_j) = U_i + V_j$), this yields $p(j \succ k \mid i) = \sigma(V_j - V_k)$ when U_i cancels.

Conceptual interpretation: The user has fixed, deterministic preferences H_{ij}, but their binary responses (clicks, likes) are noisy observations of these preferences. The randomness is in the measurement or response mechanism.

1.6.5.2 Stochastic Utility View (Random Utility Framework)

In random utility models, utilities themselves are **random variables**:

$$\tilde{H}_{ij} = f(U_i, V_j) + \varepsilon_{ij} \quad \text{(where } \varepsilon_{ij} \sim \text{Gumbel)} \tag{1.22}$$

Choices are determined by **which item achieves the highest realized utility**:

$$j \succ k \iff \tilde{H}_{ij} > \tilde{H}_{ik} \tag{1.23}$$

With Gumbel-distributed noise and additive f, this also yields $p(j \succ k \mid i) = \sigma(V_j - V_k)$ when U_i cancels.

Conceptual interpretation: The user's utility evaluation for item j fluctuates randomly across decision instances due to context, mood, unmeasured factors, or cognitive noise. The randomness is in the utility evaluation itself, not the response mechanism.

1.6.5.3 Why Both Views are Equivalent

For **pairwise choice probabilities**, both frameworks predict:

$$p(j \succ k \mid i) = \sigma(V_j - V_k) \tag{1.24}$$

They are **observationally equivalent**—given only pairwise comparison data, we cannot distinguish between them. The difference is purely conceptual:

- **Latent variable view**: Suited when we have explicit binary responses Y_{ij} (clicks, ratings) and want to model user heterogeneity explicitly
- **Random utility view**: Suited when we observe direct choices/rankings and want to model utility fluctuations

Neither view is "better"—they are **complementary perspectives** from different research traditions: - **Latent variable/Psychometrics tradition**: Measurement theory, test reliability, explicit binary responses - **Random utility/Economics tradition**: Discrete choice theory, revealed preferences, direct choices

The random utility framework we develop next provides powerful theoretical tools (IIA, Gumbel maximum properties) that explain **why** the softmax/Bradley-Terry form emerges from first principles.

1.7 Stochastic Utility Models

We now turn to a powerful mathematical framework for understanding stochastic preferences: **stochastic utility models** (also called random utility models). This framework provides the theoretical foundation for preference-based methods across machine learning—from collaborative filtering in recommender systems to RLHF in language models. As we will see, the Bradley-Terry model emerges naturally from stochastic utility models with specific noise assumptions.

An equivalent way to represent random preferences is to identify a sample $\prec$ with a vector of utilities $H = (H_j)_{j=1}^M \in \mathbb{R}^M$ where $j \succ j'$ if and only if $H_j > H_{j'}$. (For the concerned reader: We assume that $H_j = H_{j'}$ happens with zero probability; and for discrete item sets such a vector always exists.)

To get a sense for different random utility models, we consider a particular model that has the complexity of many models in modern machine learning: The Ackley function. In this model, each alternative is represented by a d-dimensional vector $(x_1, \dots, x_d) \in \mathbb{R}^d$, the Ackley function is given by

$$\text{Ackley}(x_1, x_2, \dots, x_d) = -ae^{-b\sqrt{\frac{1}{d}\sum_{j=1}^d x_j^2}} - e^{\frac{1}{d}\sum_{j=1}^d \cos(cx_j)} + a + e. \tag{1.25}$$

for some constants $a, b, c \in \mathbb{R}$. By stacking a number k of human preferences, we can compute for k samples from a random model the function in a vectorized way.

```
#| autorun: true
def ackley(X, a=20, b=0.2, c=2*np.pi):
    """
    Compute the Ackley function.
    Parameters:
      X: A NumPy array of shape (n, d) where each row is a n-dimensional point.
      a, b, c: Parameters of the Ackley function.
    Returns:
      A NumPy array of shape (n,) of function values
    """
    X = np.atleast_2d(X)
    d = X.shape[1]
    sum_sq = np.sum(X ** 2, axis=1)
    term1 = -a * np.exp(-b * np.sqrt(sum_sq / d))
    term2 = -np.exp(np.sum(np.cos(c * X), axis=1) / d)
    return term1 + term2 + a + np.e
```

We can think of the rows of X as features of different alternatives. We can visualize the full landscape of the utility function for two alternatives ($n = 2$) and features of a single dimension ($d = 1$).

```
#| autorun: true
from matplotlib.colors import LinearSegmentedColormap
ccmap = LinearSegmentedColormap.from_list("ackley", ["#f76a05", "#FFF2C9"])

def draw_surface(show=True):
    """Draw the Ackley function surface. If show=False, don't call plt.show()."""
    inps = np.linspace(-2, 2, 100)
    X, Y = np.meshgrid(inps, inps)
    grid = np.column_stack([X.ravel(), Y.ravel()])
    Z = ackley(grid).reshape(X.shape)

    plt.figure()
    contour = plt.contourf(X, Y, Z, 50, cmap=ccmap)
    plt.contour(X, Y, Z, levels=15, colors='black', linewidths=0.5, alpha=0.6)
    plt.colorbar(contour, label=r'$f(x)$', ticks=[0, 3, 6])
    plt.xlim(-2, 2)
    plt.ylim(-2, 2)
    plt.xticks([-2, 0, 2])
    plt.yticks([-2, 0, 2])
    plt.xlabel(r'$x_1$')
    plt.ylabel(r'$x_2$')
    if show:
        plt.show()
```

In this model, we can sample choice data when assuming a random generating model of, e.g., `np.random.randn(n, d)*0.5 + np.ones((n, d))*0.5`.

Item-wise (Accept-Reject) Sampling. One method for data collection is accept-reject sampling, where the decision-maker considers one item at a time and decides if they like it compared to an outside option. This is common in applications like recommendation systems, where accepting refers to a consumption signal.

We will use a simulation to familiarize ourselves with accept-reject sampling. On the surface below, blue and red points correspond to accept or reject points.

```
#| autorun: true
d = 2
n = 800
items = np.random.randn(n, d)*0.5 + np.ones((n, d))*0.5
utilities = ackley(items)
y = (utilities > utilities.mean())
draw_surface(show=False)  # Don't show yet - will add scatter overlay
plt.scatter(items[:, 0], items[:, 1], c=y, cmap='coolwarm', alpha=0.5)
plt.show()
```

Pairwise Comparisons. Pairwise comparisons, now used to fine-tune large language models can similarly be generated in this model.

```
#| autorun: true
n_pairs = 10000
pair_indices = np.random.randint(0, n, size=(n_pairs, 2))
# Exclude pairs where both indices are the same
mask = pair_indices[:, 0] != pair_indices[:, 1]
pair_indices = pair_indices[mask]

scores = np.zeros(n, dtype=int)
wins = utilities[pair_indices[:, 0]] > utilities[pair_indices[:, 1]]

# For pairs where the first item wins:
#   - Increase score for the first item by 1
#   - Decrease score for the second item by 1
np.add.at(scores, pair_indices[wins, 0], 1)
np.add.at(scores, pair_indices[wins, 1], -1)

# For pairs where the second item wins or it's a tie:
#   - Decrease score for the first item by 1
#   - Increase score for the second item by 1
np.add.at(scores, pair_indices[~wins, 0], -1)
np.add.at(scores, pair_indices[~wins, 1], 1)

# Determine preferred and non-preferred items based on scores
preferred = scores > 0
```

```
non_preferred = scores < 0

draw_surface(show=False)  # Don't show yet - will add scatter overlay
plt.scatter(items[preferred, 0], items[preferred, 1], c='blue', label='Preferred',
 ↪  alpha=0.5)
plt.scatter(items[non_preferred, 0], items[non_preferred, 1], c='purple',
 ↪  label='Non-preferred', alpha=0.5)
plt.legend()
plt.show()
```

Similarly, we can sample data for $(j, \mathcal{S})$ for $j \in \mathcal{S} \subseteq \{1, \dots, M\}$.

1.7.1 Mean Utilities

In the **stochastic utility framework**, we model utilities as stochastic random variables. Let $\tilde{H}_{ij}$ denote a random utility that can be decomposed as:

$$\tilde{H}_{ij} = f(U_i, V_j) + \varepsilon_{ij} \tag{1.26}$$

where $f(U_i, V_j)$ is the **mean utility** (deterministic component) and ε_{ij} is the **noise** (stochastic component), typically assumed independent across items.

For the **single-user or population-aggregated case** (dropping the user index i), this simplifies to:

$$\tilde{H}_j = V_j + \varepsilon_j \tag{1.27}$$

Parallel with latent variable models: Recall that in the deterministic utility framework (Section), we wrote $H_{ij} = f(U_i, V_j)$ as a deterministic quantity, with stochasticity entering through the observation process $Y_{ij} \sim \text{Bernoulli}(\sigma(H_{ij}))$. The key difference:

- **Deterministic utility view**: $H_{ij} = f(U_i, V_j)$ is fixed; randomness is in Bernoulli sampling of Y_{ij}
- **Stochastic utility view**: $\tilde{H}_{ij} = f(U_i, V_j) + \varepsilon_{ij}$ is random; randomness is in ε_{ij} within the utility itself

Both frameworks yield the same pairwise probabilities (when f is additive and the user parameter cancels), demonstrating their **observational equivalence**.

When users are modeled explicitly (as in Chapter 3), we return to the latent variable approach with H_{ij} depending on both user and item parameters. For the remainder of this chapter, we focus on the single-user or population-aggregated case using random utilities. There are different forms of noise possible. We will focus on a particular one (called Type-1 Extreme value), but others are also popular, for example Gaussian noise.

There are (at least) three different ways to view this noise ε_{ij} in the random utility framework:

- Either it is capturing the heterogeneity of different decision-makers—a view that is taken in the Economics field of Industrial Organization. Under this view, observing $(j, \mathcal{S})$ more frequently than $(j', \mathcal{S})$ is a sign of there being a higher number of decision-makers preferring j over j' than the other way around.
- or as errors of a decision-maker's optimization of utilities V_j. This view is endorsed in the literature on Bounded Rationality. Under this view, it cannot be directly concluded from frequent observation of $(j, \mathcal{S})$ compared to $(j', \mathcal{S})$ that j is preferred to j'. It might have been chosen in error.
- We can also view it as a belief of the designer about the preferences $(V_j)_{j=1}^M$. In this view, the posterior after observing data can be used to make claims about relative preferences.

These interpretations explain **why** one might model utilities as stochastic ($\tilde{H}_{ij} = f(U_i, V_j) + \varepsilon_{ij}$) rather than placing all randomness in the observation process ($Y_{ij} \sim \text{Bernoulli}(\sigma(H_{ij}))$ where $H_{ij} = f(U_i, V_j)$). The choice of framework depends on which conceptual interpretation best matches the application domain and available data.

The interpretation will guide our decision-making predictions in Chapters 4 and 5.

! Which Noise Interpretation Matters for Practice

The choice of noise interpretation is not merely philosophical—it determines **what conclusions can be drawn from data**. Under the *heterogeneity* view, observing that item j is chosen more often than j' implies a majority of the population prefers j. Under the *bounded rationality* view, the same observation may reflect systematic decision errors rather than genuine preference, so one cannot directly conclude that j is truly preferred. Under the *designer belief* view, the data updates a prior, and conclusions are conditional on the prior specification. When building systems that act on learned preferences (Chapters 4–6), the interpretation determines whether the system should optimize for the most-chosen item (heterogeneity), correct for decision errors (bounded rationality), or maintain uncertainty (Bayesian designer).

We next introduce a main way to simplify learning utility functions: The axiom of Independence of Irrelevant Alternatives.

1.8 Independence of Irrelevant Alternatives

In later chapters, we will consider cases where we sample from the most preferred elements from all items, which we call the generation task. A simple assumption will allow us to recover the probabilities of choosing any item, and in fact, the full distribution of $\prec$ from binary comparisons: The so-called Independence of Irrelevant Alternatives, introduced in Luce et al. (1959). This assumption not only allows us to easily identify a preference model, it will also massively reduce what is needed to be estimated from data: Instead of the full $(M! - 1)$-dimensional object, it will be sufficient to learn M values.

IIA assumes that the relative likelihood of choosing j compared to k does not change whether a third alternative ℓ is in the choice set or not. Formally, for every $\mathcal{S} \subseteq \{1, \dots, M\}$, $j, k \in \mathcal{S}$,

and $\ell \notin \mathcal{S}$,

$$\frac{p(j \mid \mathcal{S})}{p(k \mid \mathcal{S})} = \frac{p(j \mid \mathcal{S} \cup \{\ell\})}{p(k \mid \mathcal{S} \cup \{\ell\})}. \tag{1.28}$$

(In particular, it must be that $p(k \mid \mathcal{S}) \neq 0$ and $p(k \mid \mathcal{S} \cup \{\ell\}) \neq 0$.) That is, the relative probability of choosing j over k should be independent of whether ℓ is present in the choice set. We will show that this single assumption is sufficient to make the choice model M-dimensional, making learning feasible.

First, to our primary example: All random utility models with *independent and identically distributed* noise terms satisfy IIA. (We ask the reader to convince themselves that the Ackley function does not satisfy IIA.)

Theorem 1 (IIA-Gumbel Equivalence)

A random utility model H_j satisfies IIA if and only if we can write it as $H_j = V_j + \varepsilon_j$, where V_j is deterministic and ε_j is sampled independently and identically from the Gumbel distribution. The Gumbel distribution has cumulative distribution function $F(x) = e^{-e^{-x}}$.

Proof Sketch

( ) **Gumbel implies IIA.** If ε_j are i.i.d. Gumbel, then for any choice set $\mathcal{S}$ and any $j, k \in \mathcal{S}$:

$$\frac{p(j \mid \mathcal{S})}{p(k \mid \mathcal{S})} = \frac{e^{V_j} / \sum_{\ell \in \mathcal{S}} e^{V_\ell}}{e^{V_k} / \sum_{\ell \in \mathcal{S}} e^{V_\ell}} = \frac{e^{V_j}}{e^{V_k}} = e^{V_j - V_k} \tag{1.29}$$

This ratio depends only on $V_j - V_k$ and is independent of other alternatives in $\mathcal{S}$, so IIA holds.

( ) **IIA implies Gumbel.** The converse is more subtle: one must show that IIA together with regularity (adding alternatives cannot increase choice probability) uniquely determines the Gumbel distribution. The key insight is that IIA forces a multiplicative structure on choice probabilities, and the only continuous distributions compatible with this structure are the extreme value distributions. See Yellott Jr (1977) for the complete proof.

Gumbel vs. Gaussian Noise: Logit vs. Probit

The IIA-Gumbel equivalence shows that *only* Gumbel noise produces IIA-satisfying models (logit/softmax). Gaussian noise yields the **probit** model, which does *not* satisfy IIA—the relative probability of choosing j over k can change when a third alternative ℓ is added. In practice, the logit and probit models produce nearly identical predictions for binary comparisons, since the logistic and normal CDFs are close (after rescaling). The difference matters primarily for **multi-way choices**: the probit model allows for correlated utilities across alternatives (via a covariance matrix on ε), enabling richer substitution patterns. For most preference learning applications with pairwise data, the choice between logit and probit is empirically inconsequential; for choice sets with many similar alternatives, the probit's flexibility can be important.

Connection to Latent Variable Models

How does IIA relate to the deterministic utility (Bernoulli sampling) view?

Recall that we showed two equivalent frameworks for pairwise choice: the deterministic utility view ($H_{ij} = f(U_i, V_j)$ with Bernoulli observations) and the stochastic utility view ($\tilde{H}_{ij} = f(U_i, V_j) + \varepsilon_{ij}$ with Gumbel noise).

The connection to IIA is subtle:

- **Latent variable models** ($H_{ij} = f(U_i, V_j)$, $Y_{ij} \sim \text{Bernoulli}(\sigma(H_{ij}))$) do not inherently assume IIA. Whether IIA holds depends on the functional form f and how choices are generated from the binary responses Y_{ij}.
- **Random utility models with i.i.d. Gumbel noise** ($\tilde{H}_{ij} = f(U_i, V_j) + \varepsilon_{ij}$) automatically satisfy IIA because the noise is i.i.d. across items.

For **pairwise comparisons** from a single user with additive $f(U_i, V_j) = U_i + V_j$, both frameworks yield the same Bradley-Terry probabilities $p(j \succ k \mid i) = \sigma(V_j - V_k)$, which trivially satisfies IIA (the ratio $p(j \succ k)/p(k \succ j) = e^{2(V_j - V_k)}$ is constant).

However, for **multi-way choices** (choosing from sets $\mathcal{S}$), the frameworks can differ: - The Gumbel view yields softmax: $p(j \mid \mathcal{S}) = e^{V_j} / \sum_{k \in \mathcal{S}} e^{V_k}$ (IIA holds by construction) - The Bernoulli view requires additional assumptions about how binary responses aggregate into multi-way choices

Thus, IIA is most naturally associated with the random utility framework, though both frameworks are equivalent for pairwise data.

This is quite strong, and an **equivalence**. If we are willing to assume IIA, it is sufficient to learn M parameters to characterize the full distribution—an exponential decrease in parameters to learn. The Gumbel model may be unusual in particular for those with a stronger background in machine learning. A more familiar formulation arises for the probabilities of choice.

Theorem 2 (Choice Probabilities under IIA)

Assume a random preference model satisfies IIA, hence $H_j = V_j + \varepsilon_j$ with i.i.d. Gumbel noise. Then, the probabilities of lists are:

$$p(j_1 \succ j_2 \succ \cdots \succ j_M) = \frac{e^{V_{j_1}}}{\sum_{m=1}^{M} e^{V_{j_m}}} \cdot \frac{e^{V_{j_2}}}{\sum_{m=2}^{M} e^{V_{j_m}}} \cdots \frac{e^{V_{j_{M-1}}}}{e^{V_{j_{M-1}}} + e^{V_{j_M}}}. \tag{1.30}$$

For choices from sets,

$$p(j \mid \mathcal{S}) = \frac{e^{V_j}}{\sum_{k \in \mathcal{S}} e^{V_k}} = \text{softmax}_j((V_k)_{k \in \mathcal{S}}). \tag{1.31}$$

In particular, for binary comparisons

$$p(Y_{jj'} = 1) = \frac{e^{V_j}}{e^{V_j} + e^{V_{j'}}} = \frac{1}{1 + e^{-(V_j - V_{j'})}} = \sigma(V_j - V_{j'}). \tag{1.32}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function.

PROOF OF CHOICE PROBABILITY FORMULA

We derive (). Item j is chosen from $\mathcal{S}$ if $H_j > H_k$ for all $k \in \mathcal{S} \setminus \{j\}$, i.e., if $V_j + \varepsilon_j > V_k + \varepsilon_k$ for all $k \neq j$.

Conditioning on $\varepsilon_j = t$, we need $\varepsilon_k < V_j - V_k + t$ for all $k \neq j$. Since the ε_k are independent Gumbel with CDF $F(x) = e^{-e^{-x}}$:

$$p(j \text{ wins} \mid \varepsilon_j = t) = \prod_{k \in \mathcal{S}, k \neq j} F(V_j - V_k + t) = \prod_{k \neq j} e^{-e^{-(V_j - V_k + t)}} \tag{1.33}$$

The Gumbel density is $f(t) = e^{-t} e^{-e^{-t}}$. Integrating over t:

$$p(j \mid \mathcal{S}) = \int_{-\infty}^{\infty} \prod_{k \neq j} e^{-e^{-(V_j - V_k + t)}} \cdot e^{-t} e^{-e^{-t}} \, dt \tag{1.34}$$

Let $u = e^{-t}$, so $du = -e^{-t} dt$. After substitution and simplification (Exercise 2 asks you to complete this), we obtain:

$$p(j \mid \mathcal{S}) = \frac{e^{V_j}}{\sum_{k \in \mathcal{S}} e^{V_k}} \tag{1.35}$$

In particular, the choice probabilities $p(j \mid \mathcal{S})$ are equivalent to the multi-class logistic regression model (also called multinomial logit model), and generation from this model, that is, sampling j with probability $p(j \mid \{1, \dots, M\})$ is given by softmax-sampling $\text{softmax}((V_j)_{j=1}^M)$.

This model has many names, depending on the feedback type we consider. For binary comparison data, it is the Bradley-Terry model Bradley and Terry (1952). If data is in forms of list, it is called the Plackett-Luce model Plackett (1975). For accept-reject sampling it is also called logistic regression. For choices from subsets, it is called the (discrete choice) logit model. We will usually call the model the **logit model** and specify the feedback type.

EXAMPLE: BRADLEY-TERRY PROBABILITIES

Consider three items with utilities $V = (0, 1, 2)$. The Bradley-Terry pairwise probabilities are:

Comparison	Probability	Calculation
$p(1 \succ 2)$	0.27	$\sigma(0 - 1) = 1/(1 + e^{1}) \approx 0.27$
$p(2 \succ 1)$	0.73	$\sigma(1 - 0) = 1/(1 + e^{-1}) \approx 0.73$
$p(1 \succ 3)$	0.12	$\sigma(0 - 2) = 1/(1 + e^{2}) \approx 0.12$
$p(2 \succ 3)$	0.27	$\sigma(1 - 2) = 1/(1 + e^{1}) \approx 0.27$
$p(3 \succ 1)$	0.88	$\sigma(2 - 0) = 1/(1 + e^{-2}) \approx 0.88$
$p(3 \succ 2)$	0.73	$\sigma(2 - 1) = 1/(1 + e^{-1}) \approx 0.73$

For choice from the full set $\{1, 2, 3\}$, softmax gives:

$$p(j \mid \{1, 2, 3\}) = \frac{e^{V_j}}{e^0 + e^1 + e^2} = \frac{e^{V_j}}{1 + 2.72 + 7.39} \approx (0.09, 0.24, 0.67) \tag{1.36}$$

Item 3 (highest utility) is most likely to be chosen. Note that these utilities are only identified up to a constant: $(0, 1, 2)$ and $(5, 6, 7)$ yield identical choice probabilities.

CONNECTION TO DPO

We can now see why DPO assumes the Bradley-Terry model (). When we posit that human preferences arise from comparing implicit rewards with i.i.d. Gumbel noise—that is, a human prefers response y over y' when $r(x, y) + \varepsilon > r(x, y') + \varepsilon'$—the resulting preference probability is exactly Bradley-Terry: $p(y \succ y' \mid x) = \sigma(r(x, y) - r(x, y'))$. DPO's loss function is then the negative log-likelihood under this model.

Numerical example. Suppose for a prompt x, the model assigns:

- $\log \pi_\theta(y_w \mid x) - \log \pi_{\text{ref}}(y_w \mid x) = 0.5$ (preferred response upweighted)
- $\log \pi_\theta(y_l \mid x) - \log \pi_{\text{ref}}(y_l \mid x) = -0.3$ (dispreferred response downweighted)

With $\beta = 1$, the implicit reward difference is $0.5 - (-0.3) = 0.8$, giving preference probability $\sigma(0.8) \approx 0.69$. Under Bradley-Terry, this datapoint contributes $-\log(0.69) \approx 0.37$ to the loss.

What if Bradley-Terry fails? If human preferences exhibit:

- *Context effects*: preferences depend on what other options are shown → IIA violated
- *Intransitive preferences*: $A \succ B \succ C \succ A$ → no consistent utility exists
- *Annotator disagreement*: different people have different preferences → mixture model needed

DPO will learn a "compromise" policy that may not match any individual's true preferences. See Chapter 6 for approaches to handling heterogeneity.

The IIA assumption has many desirable properties, such as stochastic transitivity and relativity. The reader is asked to prove them in the exercises to this chapter. The learning of, and optimization based on, the mean utilities $(V_j)_{j=1}^M$ is one of the central goals of this book. Supervised learning based on it will be covered in the next chapter.

IIA is surprisingly strong, but does not allow for choice probabilities that are, fundamentally, results of multiple decision-makers making choices together. A first restriction is given by

Heterogeneity. A crucial shortcoming of IIA is that if sub-populations satisfy IIA this does not mean that the full population satisfies IIA. Assume that our population consists of sub-populations $i = 1, \dots, N$ which have mass $\alpha_1, \alpha_2, \dots, \alpha_N$, respectively, in the population, and that each of the groups has preferences satisfying IIA. Because of IIA, we can represent each sub-group's stochastic preferences with an average utility vector $(V_j^{(i)})_{j=1}^M$ for group i. The distribution of the full population is then given by a mixture of the sub-population preferences. For example, for binary comparisons

$$p(Y_{jj'} = 1) = \sum_{i=1}^{N} \alpha_i \, p(Y_{jj'} = 1 \mid \text{group } i) = \sum_{i=1}^{N} \alpha_i \, \sigma(V_j^{(i)} - V_{j'}^{(i)}). \tag{1.37}$$

Sadly, such mixtures are far from IIA, as we see in the following coding example.

```
#| autorun: true
# Define utilities for each group
group1_utilities = {'A': 1.0, 'B': 2.0, 'C': 3.0}
group2_utilities = {'A': 3.0, 'B': 2.0, 'C': 1.0}

# Group weights
alpha1 = 0.5
alpha2 = 0.5

def softmax(utilities):
    exp_vals = np.exp(list(utilities.values()))
    total = np.sum(exp_vals)
    return {k: np.exp(v) / total for k, v in utilities.items()}

# Compute mixed logit probabilities over all 3 options
p1_full = softmax(group1_utilities)
p2_full = softmax(group2_utilities)
p_mix_full = {k: alpha1 * p1_full[k] + alpha2 * p2_full[k] for k in group1_utilities}

# Compute ratio A/B in full model
ratio_full = p_mix_full['A'] / p_mix_full['B']

# Now remove option C
reduced_utils1 = {'A': group1_utilities['A'], 'B': group1_utilities['B']}
reduced_utils2 = {'A': group2_utilities['A'], 'B': group2_utilities['B']}
p1_reduced = softmax(reduced_utils1)
p2_reduced = softmax(reduced_utils2)
p_mix_reduced = {k: alpha1 * p1_reduced[k] + alpha2 * p2_reduced[k] for k in
 ↪  reduced_utils1}

# Compute ratio A/B in reduced model
ratio_reduced = p_mix_reduced['A'] / p_mix_reduced['B']

# Show violation of IIA
print(f"Ratio A/B with all options: {ratio_full:.4f}")
print(f"Ratio A/B after removing C: {ratio_reduced:.4f}")
```

An intuition for IIA's failure comes from viewing it as random utility functions: A mixture of vectors that have independent entries is not Gaussian.

```
#| autorun: true
from scipy.stats import norm

# Define two Gaussian distributions
x = np.linspace(-4, 8, 1000)
pdf1 = norm.pdf(x, 0, 1)
pdf2 = norm.pdf(x, 4, 1)

# Mixture: equal weight
mixture_pdf = 0.5 * pdf1 + 0.5 * pdf2

# Plot
plt.figure()
plt.plot(x, pdf1, label=r'$\mathcal{N}(0, 1)$', linestyle='--')
plt.plot(x, pdf2, label=r'$\mathcal{N}(4, 1)$', linestyle='--')
plt.plot(x, mixture_pdf, label='Mixture', linewidth=2)
plt.xlabel(r'$x$')
plt.ylabel(r'$p(x)$')
plt.legend()
plt.show()
```

Classic ways to solve this concern is to consider a model with explicit representation of heterogeneity, where preferences depend on a latent type $\alpha \sim F$ for some distribution F, and $(V_j)_{j=1}^M$ given α follows a random utility model with independent error terms. For example, consider a logit random utility model

$$V_j = \beta^\top x_j + \varepsilon_j \tag{1.38}$$

and assume $\beta \sim N(\mu, \Sigma)$ is a normally distributed vector, a model called the random coefficients logit model. Equivalently, we can view this as a model with correlated utility shocks.

A second limitation is only relevant if we move beyond binary choices, or observe preference lists. Let $\{1, 2, 3\}$ be three items, where items 1 and 2 are (almost) identical and different from item 3. (In the classical example, items $1, 2$ are red and blue buses, respectively, and item 3 is a train). Assume an IIA model given by average utilities (V_j). As items 1 and 2 are almost identical, assume $V_1 = V_2$. We have $p(3 \mid \{1, 3\}) = p(3 \mid \{2, 3\})$. How do these values compare to $p(3 \mid \{1, 2, 3\})$? It would be intuitive to think that item 3 is chosen with the same frequency, as there should not be more "demand" for item 3 only because item 1 is cloned. This is not the case.

```
#| autorun: true
# Deterministic utilities
v_train = 1.0
v_bus = 2.0  # Initially a single bus alternative

# Logit choice probabilities (before splitting bus)
def softmax(utilities: np.ndarray) -> np.ndarray:
```

```
    exp_util = np.exp(utilities)
    return exp_util / np.sum(exp_util)

# Before splitting: two alternatives
utilities_before = np.array([v_train, v_bus])
probs_before = softmax(utilities_before)
print("Before splitting (Train, Bus):", probs_before)

# After splitting: three alternatives
v_red_bus = v_bus
v_blue_bus = v_bus
utilities_after = np.array([v_train, v_red_bus, v_blue_bus])
probs_after = softmax(utilities_after)
print("After splitting (Train, Red Bus, Blue Bus):", probs_after)
print("After splitting, total bus share:", probs_after[1] + probs_after[2])
```

The choice probability of `train` is reduced. Why is our intuition making us think that items 1 and 2 should split their choice probability? One option is because we assume some correlation: If you like item 1 over item 3 then you should also like item 2 over item 3, and vice versa. Hence, we would like correlation between choice probabilities in random utility models. For example, if we allow in a logit random utility model the error terms for items $1, 2$ to be perfectly correlated (and we break ties uniformly at random), then

$$(p(1 \mid \{1,2,3\}), p(2 \mid \{1,2,3\}), p(3 \mid \{1,2,3\})) = \left(\frac{p(1 \mid \{1,3\})}{2}, \frac{p(2 \mid \{2,3\})}{2}, p(3 \mid \{1,3\})\right), \tag{1.39}$$

confirming our intuition.

IIA has limitations, which might require more flexible specifications of noise and heterogeneity.

> **! KEY TAKEAWAY: IIA AND BRADLEY-TERRY**
>
> The Independence of Irrelevant Alternatives (IIA) is equivalent to i.i.d. Gumbel noise in random utility models. Under IIA, choice probabilities take the softmax form, and pairwise comparisons follow the Bradley-Terry model: $p(j \succ k) = \sigma(V_j - V_k)$. This reduces parameter complexity from $M! - 1$ to just M, making learning tractable. However, IIA can fail when alternatives are similar (red-bus/blue-bus) or when the population is heterogeneous.

This is the end of our discussion of the Independence of Irrelevant Alternatives. Additional features can be found in (Train 2009; Ben-Akiva and Lerman 1985; McFadden 1981) and the original paper for logit analysis McFadden (1972).

COMMON PITFALL: FORGETTING IDENTIFIABILITY CONSTRAINTS

When fitting utility models to choice data, utilities are only identified **up to a constant**: adding the same value c to all V_j produces identical choice probabilities. A common mistake is to estimate all parameters freely, which causes non-identifiable likelihoods and numerical instability during optimization. **Always anchor one parameter** (e.g., set $V_1 = 0$) or impose a sum-to-zero constraint before fitting. The same issue arises in two-sided models (user U_i and item V_j): you cannot estimate both scales simultaneously without a normalization.

1.9 Identification and the Rashomon Effect

When learning utility parameters from choice data, two fundamental challenges arise: the **identification problem** and the **Rashomon effect**. Both concern the multiplicity of models that can explain the same observations, but they operate at different levels. These challenges apply to **both deterministic and stochastic utility models**.

The Identification Problem. A fundamental issue in utility-based models is **identification**: different utility vectors can generate identical choice probabilities or response patterns.

PROPOSITION (IDENTIFICATION)

For stochastic utility models (IIA/softmax): For any constant $c \in \mathbb{R}$, the mean utilities $(V_1, \dots, V_M)$ and $(V_1 + c, \dots, V_M + c)$ generate the same choice probabilities from $p(j \mid \mathcal{S}) = e^{V_j} / \sum_{k \in \mathcal{S}} e^{V_k}$.

For deterministic utility models: For any constant $c \in \mathbb{R}$, when utilities depend only on differences (e.g., pairwise comparisons where $p(j \succ k \mid i) = \sigma(H_{ij} - H_{ik})$), the utilities $(H_{i1}, \dots, H_{iM})$ and $(H_{i1} + c, \dots, H_{iM} + c)$ generate the same probabilities for user i.

Proof.

Stochastic case: For any $\mathcal{S}$ and $j \in \mathcal{S}$:

$$\frac{e^{V_j + c}}{\sum_{k \in \mathcal{S}} e^{V_k + c}} = \frac{e^c \cdot e^{V_j}}{e^c \cdot \sum_{k \in \mathcal{S}} e^{V_k}} = \frac{e^{V_j}}{\sum_{k \in \mathcal{S}} e^{V_k}} \tag{1.40}$$

Deterministic case: For pairwise comparisons:

$$p(j \succ k \mid i) = \sigma((H_{ij} + c) - (H_{ik} + c)) = \sigma(H_{ij} - H_{ik}) \tag{1.41}$$

This shows that choice/comparison data only identifies utility **differences**, not absolute levels. The identification problem is a structural property of both frameworks—it doesn't matter how much data we collect; we can never distinguish between (V_1, V_2, V_3) and $(V_1 + 10, V_2 + 10, V_3 + 10)$ from choice behavior alone, nor between (H_{i1}, H_{i2}, H_{i3}) and $(H_{i1} + 10, H_{i2} + 10, H_{i3} + 10)$ from pairwise comparisons alone.

Implications:

1. **Normalization required**: When learning utilities, we must fix one value. The standard convention is $V_0 = 0$ for an outside option, making all other utilities interpretable as "value relative to opting out."
2. **Only differences matter**: From choice data, we can only identify utility *differences* $V_j - V_k$, not absolute levels. This is why Bradley-Terry uses $\sigma(V_j - V_{j'})$.
3. **Connection to DPO**: In DPO, the reference policy π_{ref} provides the normalization. The implicit reward $r^*(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ measures deviation from the reference, avoiding the identification problem.

The Rashomon Effect. Even after imposing identification constraints, there may exist many *structurally different* models that fit the data equally well. This phenomenon, named after Akira Kurosawa's 1950 film *Rashomon* (where multiple witnesses give contradictory but internally consistent accounts of the same event), is called the **Rashomon effect** (Breiman 2001; Rudin et al. 2022).

Consider these examples:

- **Model class multiplicity**: A mixture of two Bradley-Terry models with group-specific utilities $(V_j^{(1)}, V_j^{(2)})$ might predict equally well as a single Bradley-Terry model with different utilities V_j, or even a non-IIA model with correlated noise.
- **Representation multiplicity**: In factor models with K dimensions, many different $(K \times M)$ embedding matrices can generate similar predicted preferences through different geometric arrangements of items.
- **Feature multiplicity**: In contextual preference models (Chapter 5), many different feature weightings θ might achieve similar prediction accuracy but assign different importance to features.

The **Rashomon ratio** quantifies this multiplicity: if the best model achieves loss L^*, the Rashomon set $\mathcal{R}_\epsilon$ contains all models with loss at most $(1 + \epsilon)L^*$. When $|\mathcal{R}_\epsilon|$ is large, many explanations coexist (Semenova, Rudin, and Parr 2022).

Why This Matters:

1. **Interpretation uncertainty**: If many models fit equally well, interpreting any single model's parameters as "the truth" is suspect. Which utility values are the "real" ones?
2. **Scientific understanding**: The Rashomon effect suggests we should report *sets* of good models rather than a single "best" model, especially for scientific inference (Breiman 2001).
3. **Alignment implications**: In RLHF, if many reward functions explain human feedback equally well, which one should we optimize? Different reward functions in the Rashomon set may induce different policies, even if they agree on the training comparisons (Skalse et al. 2022).

Relationship Between Identification and Rashomon:

- **Identification** is about the model's mathematical structure: parameters that are *algebraically equivalent* within a single model class.
- **Rashomon effect** is about empirical indistinguishability: *different models or parameterizations* that happen to fit the observed data equally well.

Think of identification as "these parameters mean the same thing," while Rashomon says "these different models perform the same."

> **EXAMPLE: RASHOMON IN PRACTICE**
>
> Suppose we collect 100 pairwise comparisons between 5 items and fit:
>
> 1. A Bradley-Terry model with utilities $(0, V_2, V_3, V_4, V_5)$ achieving 90% accuracy
> 2. A 2-group mixture model with group utilities and mixing weights, also 90% accuracy
> 3. A nested logit model with correlation parameters, also 90% accuracy
>
> All three models satisfy different identification constraints (so parameters within each model are well-defined), yet all three fit equally well. This is the Rashomon effect. Without additional data or prior knowledge, we cannot determine which model is "correct."

In later chapters, we'll see how regularization (Chapter 3), Bayesian approaches, and structural assumptions can help navigate both identification and Rashomon challenges. For now, remember: **be cautious when interpreting learned utilities as ground truth**—there may be many equally valid explanations.

1.10 Connecting Item-wise and Pairwise Models

We have now seen two families of preference models:

- **Pairwise models** (Bradley-Terry, Plackett-Luce): Model comparisons between items using item parameters V_j only
- **Item-wise models** (Rasch, factor models): Model individual responses using both user parameters U_i and item parameters V_j

How do these relate? This section clarifies the connections and provides guidance on when to use each.

From Item-wise to Pairwise. Given a factor model for item-wise data, we can derive pairwise preferences by comparing latent utilities:

Rasch ▫ Bradley-Terry (single user): As shown in , if $H_{ij} = U_i + V_j$, then $p(j \succ k \mid i) = \sigma(V_j - V_k)$, which is exactly Bradley-Terry. The user parameter cancels.

General Factor Model ▫ User-specific Preferences: If $H_{ij} = U_i^\top V_j + Z_j$, then:

$$p(j \succ k \mid i) = \sigma\left(U_i^\top (V_j - V_k) + (Z_j - Z_k)\right) \tag{1.42}$$

This is **not** standard Bradley-Terry—the preference probability depends on the user U_i! Different users may rank the same items differently. This captures the reality that a horror fan and a comedy fan will have opposite preferences between *The Shining* and *The Hangover*.

When User Parameters Cancel. User parameters cancel out in pairwise comparisons when:

1. **Rasch model**: The additive structure $H_{ij} = U_i + V_j$ means user appetite scales all items equally
2. **Single user**: With $N = 1$, there's only one U_i, which becomes part of the constant
3. **Homogeneous population**: When all users are identical ($U_i = U$ for all i)

In these cases, Bradley-Terry suffices for ranking items. But for personalized recommendations, we need factor models that preserve user information.

A Unified Generative View. Both model families arise from the same latent utility framework:

```
    Latent Utility: H_{ij} = f(U_i, V_j) +  _{ij}
                   /                        \
Item-wise Response:                  Pairwise Comparison:
Y_{ij} =  [H_{ij} > 0]               Y_{i,jk} =  [H_{ij} >
H_{ik}]
```

The choice between data types depends on the application:

Consideration	Item-wise Favored	Pairwise Favored
Data abundance	Every interaction is one datapoint	Comparisons require explicit elicitation
User identity	Users are logged in / identifiable	Users are anonymous or pooled
Goal	Personalized recommendations	Global ranking of items
Scale	$O(NM)$ observations	$O(NM^2)$ possible comparisons
Cold start	Can use user/item features	Relies only on comparison outcomes

Practical Guidance

- Use **Bradley-Terry / pairwise models** when: ranking items for a population (LLM evaluation, sports rankings), users are anonymous, or comparisons are the natural data format (A/B testing, tournament brackets)
- Use **Factor models / item-wise models** when: building personalized recommendations, users have persistent identities, or you need to predict responses to new items
- Use **both** when: you have rich interaction data and want both global quality rankings and personalized recommendations (e.g., a streaming service ranking shows globally while personalizing the home page)

1.11 Beyond Bradley-Terry: Random Choice Models (Optional)

> **Note**
>
> This section is optional and covers advanced extensions of the Bradley-Terry framework for readers interested in the broader literature on discrete choice models.

The Bradley-Terry model underlying DPO is just one member of a rich family of *random choice models* (also called *discrete choice models*) developed primarily in economics and psychology. These models provide principled ways to relax IIA when it fails to capture observed preference patterns.

Probit models. Instead of Gumbel-distributed noise (which yields Bradley-Terry/logit), we can assume Gaussian noise: $H_j = V_j + \varepsilon_j$ with $\varepsilon \sim \mathcal{N}(0, \Sigma)$. When $\Sigma = I$ (independent noise), this yields the *multinomial probit* model. The key advantage is that probit allows for arbitrary correlation structures in Σ, naturally handling the red-bus/blue-bus problem. If red and blue buses have correlated noise ($\Sigma_{12} > 0$), adding one does not steal probability from the train. The disadvantage is computational: unlike logit, probit choice probabilities lack closed-form expressions and require numerical integration.

Nested logit. A tractable middle ground is the *nested logit* model, which groups alternatives into "nests" with correlated noise within each nest. For the transportation example, we might have Bus nest = {red bus, blue bus} and Train nest = {train}. The model first chooses a nest, then an alternative within the nest. This preserves IIA *within* nests while allowing substitution patterns *across* nests that violate IIA.

Random coefficients (mixed) logit.

The most flexible extension is the *random coefficients logit* (also called *mixed logit*), which we already encountered in the heterogeneity discussion. Here, the utility coefficients β are random:

$$V_j = \beta^\top x_j, \quad \beta \sim F \tag{1.43}$$

for some distribution F (often Gaussian). This model can approximate any random utility model arbitrarily well (McFadden and Train 2000), making it extremely flexible. Modern implementations use simulation-based estimation; see Train (2009) for details.

Gaussian Process models for nonlinear rewards. {#sec-gp-models}

The random coefficients logit still assumes utility is *linear* in features x_j. This assumption is limiting: linear reward functions can only achieve their optima at extreme points (corners of the feasible region). Consider a robot learning preferences over mini-golf shots parameterized by speed and angle—a linear reward $r = w_1 \cdot \text{speed} + w_2 \cdot \text{angle}$ cannot express "I prefer a moderate shot to the middle target."

Gaussian Processes (GPs) provide a nonparametric alternative that can model arbitrary smooth functions:

$$r(x) \sim \mathcal{GP}(m, k) \tag{1.44}$$

where $m : \mathcal{X} \to \mathbb{R}$ is the mean function and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the covariance function (kernel). A GP is defined by the property that any finite collection of function values follows a multivariate Gaussian:

$$[r(x_1), \ldots, r(x_n)]^\top \sim \mathcal{N}(\mu, K), \quad \text{where } \mu_i = m(x_i),\ K_{ij} = k(x_i, x_j) \tag{1.45}$$

The most common kernel is the Radial Basis Function (RBF):

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \tag{1.46}$$

which encodes the assumption that nearby points have correlated function values, with length-scale ℓ controlling smoothness.

```
#| autorun: true

def rbf_kernel(X1, X2, sigma_f=1.0, length_scale=1.0):
    """Compute RBF kernel matrix between two sets of points."""
    sq_dist = np.sum(X1**2, axis=1, keepdims=True) + np.sum(X2**2, axis=1) - 2 * X1 @
 ↪  X2.T
    return sigma_f**2 * np.exp(-sq_dist / (2 * length_scale**2))

# Sample from GP prior
X_grid = np.linspace(-3, 3, 100).reshape(-1, 1)
K = rbf_kernel(X_grid, X_grid, sigma_f=1.0, length_scale=1.0)
K += 1e-6 * np.eye(len(K))  # numerical stability

# Draw samples from GP prior
samples = np.random.multivariate_normal(np.zeros(len(X_grid)), K, size=5)

fig, axes = plt.subplots(1, 2, figsize=(6, 3))

# Left: Sample functions from GP prior
for i, sample in enumerate(samples):
    axes[0].plot(X_grid, sample, alpha=0.7, label=f'Sample {i+1}')
axes[0].axhline(y=0, color='k', linestyle='--', alpha=0.3)
axes[0].set_xlabel('Input x')
axes[0].set_ylabel(r'$r(x)$')
axes[0].set_title(r'Samples from GP Prior')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Right: Compare different length scales
for ls, color in [(0.3, '#1f77b4'), (1.0, '#ff7f0e'), (3.0, '#2ca02c')]:
    K_ls = rbf_kernel(X_grid, X_grid, sigma_f=1.0, length_scale=ls)
    K_ls += 1e-6 * np.eye(len(K_ls))
    sample = np.random.multivariate_normal(np.zeros(len(X_grid)), K_ls)
    axes[1].plot(X_grid, sample, color=color, alpha=0.8, label=r'$\ell$'+ f'= {ls}')
```

```
34
35 axes[1].set_xlabel('Input x')
36 axes[1].set_ylabel(r'$r(x)$')
37 axes[1].set_title('Effect of Length-scale on GP Samples')
38 axes[1].legend()
39 axes[1].grid(True, alpha=0.3)
40 plt.show()
```

When combined with the Bradley-Terry likelihood, we model preferences as $p(A \succ B \mid r) = \sigma(r(x_A) - r(x_B))$ where $r \sim \mathcal{GP}(m, k)$. Unlike standard GP regression where we observe function values directly, preference data provides only ordinal information—which item is preferred—making inference more challenging.

Why GPs for preferences? (1) *Expressiveness*: Can represent complex nonlinear functions without specifying a parametric form. (2) *Uncertainty quantification*: GP provides predictive means and confidence intervals. (3) *Active learning*: Uncertainty guides which comparisons to query (Chapter 4). (4) *Bayesian framework*: Principled posterior updates as new comparisons arrive.

Trade-offs: GPs are well-suited when the reward function has unknown nonlinear structure and the feature dimension is low-to-moderate ($d \lesssim 10$). For high-dimensional problems or large datasets, linear models or neural networks may be more practical due to the $O(n^3)$ computational cost of GP inference. Chapter 3 covers inference techniques (including Laplace approximation for non-Gaussian likelihoods), and Chapter 4 shows how GP uncertainty enables sample-efficient active query selection.

Implications for preference learning. For practitioners using DPO or RLHF, these extensions suggest directions for improvement:

1. **Heterogeneous populations**: If annotators have diverse preferences, a single Bradley-Terry model may be misspecified. Random coefficients models (or mixture models) can capture this heterogeneity.
2. **Correlated alternatives**: When comparing responses that differ only slightly (e.g., same content with different formatting), IIA violations may occur. Nested or probit models can help.
3. **Choice set effects**: If preferences depend on what alternatives are present, IIA fails. This is particularly relevant for k-wise comparisons with $k > 2$.

These extensions are active areas of research in machine learning from human preferences; see Siththaranjan, Laidlaw, and Hadfield-Menell (2023) for recent work on distributional preference learning that relaxes Bradley-Terry assumptions.

The next chapter is the first to study learning of average utility functions from preference data, and assumes that a dataset is given of utility vectors $(V_j)_{j=1}^M$ for different types of sampling and for different notions of "inference".

1.12 Summary

- Preference data pervades machine learning—from recommender systems and information retrieval to robotics and language model alignment—and this chapter provides the unified probabilistic foundation for modeling it.
- **Deterministic** utility models assign a fixed score V_j to each item but cannot explain why the same person sometimes chooses differently; **stochastic** (random utility) models add noise ε_j to capture this variability.
- The **Rasch model** decomposes item-wise response probabilities into user appetite U_i and item appeal V_j via $p(Y_{ij} = 1) = \sigma(U_i + V_j)$. The **Bradley-Terry model** for pairwise comparisons emerges as a special case: $p(j \succ k) = \sigma(V_j - V_k)$.
- **Independence of Irrelevant Alternatives (IIA)** is the key structural assumption: it makes estimation tractable (only M parameters) but fails under population heterogeneity and the red-bus/blue-bus cloning problem.
- **K-dimensional factor models** extend scalar utilities to vector embeddings $U_i, V_j \in \mathbb{R}^K$, enabling richer representations needed for recommender systems and multi-dimensional preferences.
- **Gaussian Processes** provide a nonparametric alternative when the utility function's form is unknown, placing a prior directly over functions rather than assuming a parametric structure.
- The **Bradley-Terry model connects directly to DPO**: the DPO objective for language model alignment is equivalent to Bradley-Terry maximum likelihood with utilities defined by the policy's log-ratio.
- **Plackett-Luce** generalizes Bradley-Terry from pairwise comparisons to full rankings, and all these models share the same random utility foundation with i.i.d. Gumbel noise.

1.13 Quick Check

Test your understanding before proceeding to the exercises.

1. In the Rasch model $p(Y_{ij} = 1) = \sigma(U_i + V_j)$, why can't we estimate both U_i and V_j without a constraint? What happens if we add a constant to all U_i and subtract it from all V_j?
2. If two items have identical appeal $V_j = V_k$, what does Bradley-Terry predict about $p(j \succ k)$? What does this mean for learning from such a comparison?
3. A recommender system adds a duplicate of an already-popular item to the catalog. Under IIA, what happens to the choice probabilities of the original items? Why is this problematic? (Hint: red bus/blue bus.)
4. When would you prefer a Gaussian Process preference model over a K-dimensional factor model? What is the main computational tradeoff?

1.14 Discussion Questions

- How does modeling preferences as **random** (rather than deterministic) help us capture real-world choice behavior?
- How does the **Rasch model** differ from standard logistic regression? What additional structure does it impose by having separate user and item parameters?
- Why does **user appetite** U_i **cancel out** in pairwise comparisons under the Rasch model? When is this a feature (simplification) versus a limitation (loss of information)?
- What is the **Independence of Irrelevant Alternatives** (IIA) axiom, and why does it simplify the estimation of choice models?
- Why do i.i.d. Gumbel shocks in a random utility model lead to the Plackett–Luce (list) and softmax/logit (choice) formulas?
- In what ways can **binary comparisons**, **choice-from-a-set**, and **full rankings** each be seen as observations of the same underlying stochastic preference distribution?
- When would you prefer to collect **item-wise data** (accept/reject, ratings) versus **pairwise comparisons**? Consider cost, information content, and downstream tasks.
- How does introducing an "outside option" (item 0) allow us to interpret **accept–reject** data (e.g., clicks in a recommender system) within the same logit framework?
- Why does mixing multiple IIA-satisfying sub-populations generally **violate** IIA at the aggregate level? What does this imply for preference learning from diverse user populations?
- Explain the "**red bus–blue bus**" problem: why does splitting a single alternative into two identical ones distort logit choice probabilities? Give an example of when this might occur in recommender systems or search ranking.
- Modern recommender systems use **neural networks** to compute user and item embeddings. How does this relate to the K-dimensional factor model $H_{ij} = U_i^\top V_j + Z_j$?
- The **ideal point model** ($H_{ij} = -\|U_i - V_j\|_2 + Z_j$) is popular in political science. Why might Euclidean distance be more appropriate than dot product for modeling political preferences?
- Compare implicit preference signals (clicks, watch time, purchases) with explicit ratings. What are the tradeoffs for preference learning?
- The Elo rating system in chess uses the Bradley-Terry model. What assumptions does this make about player skill, and when might these assumptions fail?

1.15 Bibliographic Notes

Random utility models originate in Thurstone's (1927) work on psychophysical scaling, where he proposed that perceived stimuli have random "discriminal processes." The axiomatic approach via the choice axiom (equivalent to IIA) was developed by Luce et al. (1959), providing the mathematical foundations for probabilistic choice theory. Yellott Jr (1977) proved the remarkable equivalence between IIA and Gumbel-distributed noise.

The Rasch model was introduced by Rasch (1960) for psychometric testing, where U_i represents student ability and V_j represents item difficulty. The model has become foundational

in educational testing and is used in standardized tests like the GRE. For comprehensive treatment of item response theory, see Baker (2001). The connection between Rasch models and Bradley-Terry was explored in the paired comparison literature; see Andrich (1978).

Factor models for recommendations gained prominence after the Netflix Prize competition, where matrix factorization methods proved highly effective (Koren, Bell, and Volinsky 2009). The probabilistic interpretation connecting matrix factorization to latent factor models is developed in Mnih and Salakhutdinov (2007). Modern neural approaches to collaborative filtering build on these foundations; see He et al. (2017) for neural collaborative filtering.

Ideal point models originate in Coombs (1950)'s unfolding theory and were developed for political science applications by Poole and Rosenthal (1985) for analyzing roll-call voting. The connection to preference learning is explored in Carroll and Chang (1972). For modern computational approaches, see Armstrong et al. (2014).

Discrete choice econometrics was pioneered by McFadden (1974), who developed the conditional logit model and its applications to transportation choice. His subsequent work on nested logit and mixed logit models, along with his contributions to the econometric theory of discrete choice, earned him the Nobel Prize in Economics in 2000 (shared with James Heckman). The definitive graduate textbook treatment is Train (2009).

The Bradley-Terry model was introduced by Bradley and Terry (1952) for analyzing paired comparisons in ranking problems, though similar models were developed independently by Zermelo (1929) for chess ratings. The extension to partial rankings is due to Plackett (1975) and Luce et al. (1959). For modern statistical treatments, see Cattelan (2012).

Proper scoring rules have a long history in probability forecasting; see Gneiting and Raftery (2007) for a comprehensive treatment. The connection between proper scoring rules and calibration is fundamental to understanding why cross-entropy is the standard loss for classification and language modeling.

Preference learning in machine learning has grown rapidly since Christiano et al. (2017) introduced RLHF for learning from human preferences without hand-specified reward functions. Ouyang et al. (2022) scaled this approach to large language models. Rafailov et al. (2023) introduced DPO, making explicit the Bradley-Terry assumption underlying reward modeling. For connections to social choice theory, see Arrow (1951) and Sen (1986).

For further reading: On the economics of discrete choice, Ben-Akiva and Lerman (1985) provides accessible coverage. On the psychology of choice, Kahneman (2011) discusses bounded rationality and systematic deviations from utility maximization. On recent advances in preference learning for AI, see the surveys by Kaufmann et al. (2023).

Essential Reading

If you read only five papers from this chapter's references, make them these:

1. **Thurstone (1927)** — The 1927 origin of random utility models for paired comparisons

2. **Luce et al.** (1959) — The axiomatic foundations of IIA and choice theory
3. **McFadden** (1974) — The conditional logit model that launched discrete choice econometrics (Nobel Prize 2000)
4. **Bradley and Terry** (1952) — The Bradley-Terry model for paired comparison ranking
5. **Rafailov et al.** (2023) — DPO: the modern connection between Bradley-Terry and LLM alignment

1.16 Further Reading

For readers who want to go deeper into the topics introduced in this chapter, we recommend the following:

- **Train** (2009), *Discrete Choice Methods with Simulation* — The definitive graduate textbook on random utility models, covering mixed logit, simulation-based estimation, and welfare analysis. Essential for understanding the econometric foundations of preference models.
- **Luce et al.** (1959), *Individual Choice Behavior* — Luce's original monograph developing the choice axiom (IIA) from first principles. Remarkably clear and still relevant 65 years later.
- **Kahneman** (2011), *Thinking, Fast and Slow* — The essential reference on bounded rationality and systematic deviations from utility maximization, providing context for why the rational choice models in this chapter are useful approximations but not complete descriptions of human behavior.
- **Cattelan** (2012), "Models for Paired Comparison Data: A Review with Emphasis on Dependent Data" — A comprehensive survey of Bradley-Terry extensions, including models for ties, home advantage, and time-varying abilities.
- **Chu and Ghahramani** (2005), "Preference Learning with Gaussian Processes" — The foundational paper on GP-based preference learning that underpins the nonparametric models in this chapter and the active learning methods in Chapter 3.
- **McFadden** (2000), "Disaggregate Behavioral Travel Demand's RUM Side" — McFadden's own retrospective on discrete choice theory, including the limitations of IIA and the development of mixed logit as an IIA-relaxing alternative.
- **Ben-Akiva and Lerman** (1985), *Discrete Choice Analysis* — An accessible introduction to discrete choice models, emphasizing applications in transportation and marketing that motivated much of the theory.

1.17 Exercises

Exercises are marked with difficulty levels: (*) for introductory, (**) for intermediate, and (***) for challenging.

1.17.1 Properties of IIA Models (*)

Prove that if a preference model satisfies IIA, it will also satisfy $p(j \mid \mathcal{S}) \leq p(j \mid \mathcal{S}')$ for any j and $\mathcal{S} \supseteq \mathcal{S}'$ (called regularity) and for all (j, k, ℓ), if $p(j \mid \{j, k\}) \geq 0.5$ and $p(k \mid \{k, \ell\}) \geq 0.5$, then necessarily $p(j \mid \{j, \ell\}) \geq 0.5$.

1.17.2 Discrete Choice Models (**)

Consider a linear random utility model $V_j = \beta^\top x_j + \varepsilon_j$ for $j = 1, 2, \dots, M$, where ε_j is i.i.d. sampled from a Gumbel distribution. We would like to compute $p(j \mid \{1, \dots, M\})$ and connect it to multi-class logistic regression.

(a) First compute $p(H_j < t)$ for any j in terms of F. Use this probability to provide a formula for $p(j \mid \{1, \dots, M\})$ over t in terms of f and F.

(b) Compute the integral derived in part (a) with the appropriate u-substitution. You should arrive at the multi-class logistic regression model.

1.17.3 Mixtures and correlations (*)

Prove that the class of random preferences induced by the following two are identical: (a) mixtures of IIA random utility models (that is, those with i.i.d. noise) (b) random utility models with correlated noise.

1.17.4 Sufficient Statistics (***)

(a) Show that choice data completely specifies the preference model. That is, express $p(\prec)$ for any $\prec$ in terms of $p(j \mid \mathcal{S})$, $j \in \mathcal{S} \subseteq \{1, \dots, M\}$.

(b) Show that this is not the case for binary comparisons. That is, give an example of two different preference models that induce the same probabilities $p(Y_{jk} = 1)$.

Hint: For (a), consider how you would reconstruct the probability of a ranking $j_1 \succ j_2 \succ j_3$ from choice probabilities. For (b), consider M=3 and try to find two distributions over the 6 possible rankings that agree on all pairwise margins.

1.17.5 Non-Random Utility Models (***)

Not all probability assignments for binary comparisons $p_{jk} = p(Y_{jk} = 1)$ can be realized with a random preference model. Give an example of binary comparisons (p_{12}, p_{23}, p_{31}) that cannot be a result of a random preference model.

Hint: Think about what constraints transitivity imposes. If $p_{12} > 0.5$ and $p_{23} > 0.5$, what can you say about p_{13}?

1.17.6 Bradley-Terry Maximum Likelihood (**)

You observe the following pairwise comparison outcomes between 4 chess players over a tournament:

Comparison	Player A wins	Player B wins
1 vs 2	65	35
1 vs 3	72	28
1 vs 4	80	20
2 vs 3	58	42
2 vs 4	70	30
3 vs 4	55	45

(a) Write the log-likelihood for the Bradley-Terry model as a function of utilities $V = (V_1, V_2, V_3, V_4)$ with the identification constraint $V_4 = 0$.

(b) Derive the gradient of the log-likelihood. Implement gradient ascent to find the MLE.

(c) Interpret the learned utilities. Which player is strongest? What is the estimated probability that Player 1 beats Player 2?

1.17.7 From Preferences to Policy (**)

This exercise connects Bradley-Terry estimation to DPO.

(a) Show that minimizing the DPO loss () on a dataset of preferences is equivalent to maximizing the Bradley-Terry log-likelihood, where the "utility" of response y given prompt x is $\beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$.

(b) Suppose 20% of preference labels are "noisy"—i.e., the annotator flipped a coin instead of expressing their true preference. How does this affect the learned policy compared to noise-free data?

(c) If you had access to the noise rate, how would you modify the DPO objective to account for it?

1.17.8 Rasch Model Properties (*)

Consider the Rasch model $p(Y_{ij} = 1) = \sigma(U_i + V_j)$ where U_i is user appetite and V_j is item appeal.

(a) Show that the log-odds of acceptance is linear in the parameters:

$$\log \frac{p(Y_{ij} = 1)}{p(Y_{ij} = 0)} = U_i + V_j \tag{1.47}$$

(b) Prove that for a single user i comparing items j and k, the Rasch model implies Bradley-Terry:

$$p(j \succ k \mid i) = \sigma(V_j - V_k) \tag{1.48}$$

What happens to the user parameter U_i?

(c) Explain why item-wise data from N users can identify both $\{U_i\}_{i=1}^N$ and $\{V_j\}_{j=1}^M$ (up to a single additive constant), while pairwise comparison data cannot identify the U_i parameters at all.

1.17.9 Factor Model Comparison (**)

Consider two factor models with K-dimensional embeddings:

- **Dot-product model**: $H_{ij} = U_i^\top V_j$
- **Ideal-point model**: $H_{ij} = -\|U_i - V_j\|_2^2$

(a) Show that with $K = 1$, the two models can represent the same set of preference probabilities. *Hint: Find a transformation between the parameters.*

(b) Give an intuitive example of a preference pattern that the dot-product model can represent but the ideal-point model cannot. *Hint: Consider what happens when a user's embedding points in the opposite direction from an item's embedding.*

(c) In what application domains (beyond those mentioned in the chapter) would you prefer the ideal-point model over the dot-product model? Justify your answer.

1.17.10 From Rasch to Recommendations (*)

A music streaming service has 1000 users and 500 songs. Each user has listened to some songs (listen = $Y_{ij} = 1$) and skipped others when recommended (skip = $Y_{ij} = 0$). You model this with the Rasch model.

(a) Write down the log-likelihood for the Rasch model given an observed response matrix Y.

(b) After fitting the model, you estimate user appetite $U_{\text{Alice}} = 1.5$ for user Alice and item appeal $V_{\text{song}} = -0.5$ for the song "Bohemian Rhapsody". What is the probability that Alice will listen to this song if recommended?

(c) User Bob has estimated appetite $U_{\text{Bob}} = -0.5$. Without knowing the item parameters, compute the odds ratio comparing Alice's probability of listening to *any* song versus Bob's probability.

1.17.11 Posterior Inference for Mixture Preferences (**)

(This exercise previews some of the aspects for learning utility functions from the next chapter but is self-contained.) You are part of the ML team on the movie streaming site "Preferential". You receive full preference orderings in the form $j_1 \succ j_2 \succ \cdots \succ j_M$, where j_1 is the most, and j_M the least preferred option. The preferences come from 600 distinct users with 50 examples per user. Each movie j has a 10-dimensional feature vector $\mathbf{x}_j$, and each user i has a 10-dimensional weight vector $\mathbf{w}_i$. The preferences for user i follow the random utility model $V_j = \mathbf{w}_i^\top \mathbf{x}_j + \varepsilon_j$, where ε_j is i.i.d. Gumbel distributed.

Sadly, you lost all user identifiers. Unashamedly, you assume a model where a proportion p of the users have weights $\mathbf{w}_1$, and a proportion $1-p$ has weights $\mathbf{w}_2$. Each user belongs to one of two groups: users with weights $\mathbf{w}_1$ are part of Group 1, and users with weights $\mathbf{w}_2$ are part of Group 2.

(a) For a datapoint $(j_1 \succ j_2)$ and conditional on p, $\mathbf{w}_1$ and $\mathbf{w}_2$, compute the likelihood $p(Y_{j_1 j_2} = 1 \mid p, \mathbf{w}_1, \mathbf{w}_2)$.

(b) Use the likelihood to simplify the posterior distribution of $p, \mathbf{w}_1, \mathbf{w}_2$ after updating on $(\mathbf{x}_{j_1}, \mathbf{x}_{j_2})$ leaving terms for the priors unchanged.

(c) Assume priors $p \sim B(1,1)$, $w_1 \sim N(0, \mathbf{I})$, and $w_2 \sim N(0, \mathbf{I})$ where B represents the Beta distribution and $\mathcal{N}$ represents the normal distribution (all three sampled independently). You will notice that the posterior from part (b) has no simple closed form, requiring numerical methods. One such method, allowing to approximate sample from the posterior π, is called Metropolis-Hastings. (The reason why one might want to sample from the posterior will be discussed later.) Broadly, the idea of Metropolis-Hastings and similar, so-called Markov Chain Monte Carlo methods is the following: Construct a Markov chain $\{x_t\}_{t=1}^{\infty}$ which has as "ergodic" distribution given by your desired distribution.[3] By properties of Markov chains, for $t \gg 0$, x_t will be almost as good as sampled from the "ergodic" distribution. In Metropolis-Hastings, the distribution is a proposal P for x_{t+1} is made via sampling from a chosen probability kernel $Q(\bullet|x_t)$ (e.g., adding Gaussian noise). The acceptance probability of the proposal is given by

$$x_{t+1} = \begin{cases} \tilde{Q}(\bullet|x_t) & \text{with probability } A, \\ x_t & \text{with probability } 1-A. \end{cases} \tag{1.49}$$

where

$$A = \min\left(1, \frac{\pi(\bullet)Q(x_t|\bullet)}{\pi(x_t)Q(\bullet|x_t)}\right). \tag{1.50}$$

We will extract samples from the Markov chain after a "burn-in period", $(x_{T+1}, x_{T+2}, \cdots, x_N)$.

To build some intuition, suppose we have a biased coin that turns heads with probability p_{heads}. We observe 12 coin flips to have 9 heads (H) and 3 tails (T). If our prior for p_{H} was $B(1,1)$,

[3] That is, x_t is independent of $(x_1, x_2, \ldots, x_{t-2})$ conditional on x_{t-1}.

then, by properties of the Beta distribution, our posterior will be $B(1+9, 1+3) = B(10, 4)$. The Bayesian update is given by

$$p(p_{\mathrm{H}}|9\mathrm{H}, 3\mathrm{T}) = \frac{p(9\mathrm{H}, 3\mathrm{T}|p_{\mathrm{H}})B(1,1)(p_{\mathrm{H}})}{\int_0^1 P(9\mathrm{H}, 3\mathrm{T}|p_{\mathrm{H}})B(1,1)(p_{\mathrm{H}})\,\mathrm{d}p_{\mathrm{H}}} = \frac{p(9\mathrm{H}, 3\mathrm{T}|p_{\mathrm{H}})}{\int_0^1 p(9\mathrm{H}, 3\mathrm{T}|p_{\mathrm{H}})\,\mathrm{d}p_{\mathrm{H}}}. \tag{1.51}$$

Find the acceptance probability A in the setting of the biased coin assuming the proposal distribution $Q(\cdot|x_t) = x_t + N(0, \sigma)$ for given σ. Notice that this choice of Q is symmetric, i.e., $Q(x_t|p) = Q(p|x_t)$ for all $p \in \mathbb{R}$. Note that it is unnecessary to compute the normalizing constant of the Bayesian update (i.e., the integral in the denominator). This simplification is one of the main practical advantages of Metropolis-Hastings.

(d) Implement Metropolis-Hastings to sample from the posterior distribution of the biased coin in `multimodal_preferences/biased_coin.py`. Attach a histogram of your MCMC samples overlayed on top of the true posterior $B(10, 4)$ by running `python biased_coin.py`.

```
#| autorun: true
#| error: true
from scipy.stats import beta

def likelihood(p: float) -> float:
    """
    Computes the likelihood of 9 heads and 3 tails, assuming p_heads is p.

    Args:
    p (float): A value between 0 and 1 representing the probability of heads.

    Returns:
    float: The likelihood value at p_heads=p. Return 0 if p is outside the range [0,
    1].
    """
    # YOUR CODE HERE (~1-3 lines)
    pass
    # END OF YOUR CODE


def propose(x_current: float, sigma: float) -> float:
    """
    Proposes a new sample from the proposal distribution Q.
    Here, Q is a normal distribution centered at x_current with standard deviation
    sigma.

    Args:
    x_current (float): The current value in the Markov chain.
    sigma (float): Standard deviation of the normal proposal distribution.
```

```
    Returns:
    float: The proposed new sample.
    """
    # YOUR CODE HERE (~1-3 lines)
    pass
    # END OF YOUR CODE


def acceptance_probability(x_current: float, x_proposed: float) -> float:
    """
    Computes the acceptance probability A for the proposed sample.
    Since the proposal distribution is symmetric, Q cancels out.

    Args:
    x_current (float): The current value in the Markov chain.
    x_proposed (float): The proposed new value.

    Returns:
    float: The acceptance probability
    """
    # YOUR CODE HERE (~4-6 lines)
    pass
    # END OF YOUR CODE


def metropolis_hastings(N: int, T: int, x_init: float, sigma: float) -> np.ndarray:
    """
    Runs the Metropolis-Hastings algorithm to sample from a posterior distribution.

    Args:
    N (int): Total number of iterations.
    T (int): Burn-in period (number of initial samples to discard).
    x_init (float): Initial value of the chain.
    sigma (float): Standard deviation of the proposal distribution.

    Returns:
    list: Samples collected after the burn-in period.
    """
    samples = []
    x_current = x_init

    for t in range(N):
        # YOUR CODE HERE (~7-10 lines)
        # Use the propose and acceptance_probability functions to get x_{t+1} and
 ↪  store it in samples after the burn-in period T
        pass
        # END OF YOUR CODE

```

```
    return samples


def plot_results(samples: np.ndarray) -> None:
    """
    Plots the histogram of MCMC samples along with the true Beta(10, 4) PDF.

    Args:
    samples (np.ndarray): Array of samples collected from the Metropolis-Hastings algorithm.

    Returns:
    None
    """
    # Histogram of the samples from the Metropolis-Hastings algorithm
    plt.hist(samples, bins=50, density=True, alpha=0.5, label="MCMC Samples")

    # True Beta(10, 4) distribution for comparison
    p = np.linspace(0, 1, 1000)
    beta_pdf = beta.pdf(p, 10, 4)
    plt.plot(p, beta_pdf, "r-", label="Beta(10, 4) PDF")

    plt.xlabel("$p_heads$")
    plt.ylabel("Density")
    plt.title("MH Samples from Posterior")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    # MCMC Parameters (DO NOT CHANGE!)
    N = 50000  # Total number of iterations
    T = 10000  # Burn-in period to discard
    x_init = 0.5  # Initial guess for p_heads
    sigma = 0.1  # Standard deviation of the proposal distribution

    # Run Metropolis-Hastings and plot the results
    samples = metropolis_hastings(N, T, x_init, sigma)
    plot_results(samples)
```

(e) Implement Metropolis-Hastings in the movie setting inside `multimodal_preferences/movie_metropolis.py`. You should be able to achieve a 90% success rate with most `fraction_accepted` values above 0.1. Success is measured by thresholded closeness of predicted parameters to true parameters. You may notice occasional failures that occur due to lack of convergence which we will account for in grading.

```
#| autorun: true
#| error: true
import torch
import torch.distributions as dist
import math
from tqdm import tqdm
from typing import Tuple

def make_data(
    true_p: torch.Tensor, true_weights_1: torch.Tensor, true_weights_2: torch.Tensor, num_movies: int, feature_dim: int
) -> Tuple[torch.Tensor, torch.Tensor]:
    """
    Generates a synthetic movie dataset according to the CardinalStreams model.

    Args:
        true_p (torch.Tensor): Probability of coming from Group 1.
        true_weights_1 (torch.Tensor): Weights for Group 1.
        true_weights_2 (torch.Tensor): Weights for Group 2.

    Returns:
        Tuple[torch.Tensor, torch.Tensor]: A tuple containing the dataset and labels.
    """
    # Create movie features
    first_movie_features = torch.randn((num_movies, feature_dim))
    second_movie_features = torch.randn((num_movies, feature_dim))

    # Only care about difference of features for Bradley-Terry
    dataset = first_movie_features - second_movie_features

    # Get probabilities that first movie is preferred assuming Group 1 or Group 2
    weight_1_probs = torch.sigmoid(dataset @ true_weights_1)
    weight_2_probs = torch.sigmoid(dataset @ true_weights_2)

    # Probability that first movie is preferred overall can be viewed as sum of conditioning on Group 1 and Group 2
    first_movie_preferred_probs = (
        true_p * weight_1_probs + (1 - true_p) * weight_2_probs
    )
    labels = dist.Bernoulli(first_movie_preferred_probs).sample()
    return dataset, labels


def compute_likelihoods(
    dataset: torch.Tensor,
    labels: torch.Tensor,
    p: torch.Tensor,
    w_1: torch.Tensor,
    w_2: torch.Tensor,
```

```
) -> torch.Tensor:
    """
    Computes the likelihood of each datapoint. Use your calculation from part (a) to help.

    Args:
        dataset (torch.Tensor): The dataset of differences between movie features.
        labels (torch.Tensor): The labels where 1 indicates the first movie is preferred, and 0 indicates preference of the second movie.
        p (torch.Tensor): The probability of coming from Group 1.
        w_1 (torch.Tensor): Weights for Group 1.
        w_2 (torch.Tensor): Weights for Group 2.

    Returns:
        torch.Tensor: The likelihoods for each datapoint. Should have shape (dataset.shape[0], )
    """
    # YOUR CODE HERE (~6-8 lines)
    pass
    # END OF YOUR CODE

def compute_prior_density(
    p: torch.Tensor, w_1: torch.Tensor, w_2: torch.Tensor
) -> torch.Tensor:
    """
    Computes the prior density of the parameters.

    Args:
        p (torch.Tensor): The probability of preferring model 1.
        w_1 (torch.Tensor): Weights for model 1.
        w_2 (torch.Tensor): Weights for model 2.

    Returns:
        torch.Tensor: The prior densities of p, w_1, and w_2.
    """
    # Adjusts p to stay in the range [0.3, 0.7] to prevent multiple equilibria issues at p=0 and p=1
    p_prob = torch.tensor([2.5]) if 0.3 <= p <= 0.7 else torch.tensor([0.0])

    def normal_pdf(x: torch.Tensor) -> torch.Tensor:
        """Computes the PDF of the standard normal distribution at x."""
        return (1.0 / torch.sqrt(torch.tensor(2 * math.pi))) * torch.exp(-0.5 * x**2)

    weights_1_prob = normal_pdf(w_1)
    weights_2_prob = normal_pdf(w_2)

    # Concatenate the densities
    concatenated_prob = torch.cat([p_prob, weights_1_prob, weights_2_prob])
    return concatenated_prob
```

```
93
94
95  def metropolis_hastings(
96      dataset: torch.Tensor,
97      labels: torch.Tensor,
98      sigma: float = 0.01,
99      num_iters: int = 30000,
100     burn_in: int = 20000,
101 ) -> Tuple[torch.Tensor, torch.Tensor, torch.Tensor, float]:
102     """
103     Performs the Metropolis-Hastings algorithm to sample from the posterior
    ↪ distribution.
104     DO NOT CHANGE THE DEFAULT VALUES!
105
106     Args:
107         dataset (torch.Tensor): The dataset of differences between movie features.
108         labels (torch.Tensor): The labels indicating which movie is preferred.
109         sigma (float, optional): Standard deviation for proposal distribution.
110             Defaults to 0.01.
111         num_iters (int, optional): Total number of iterations. Defaults to 30000.
112         burn_in (int, optional): Number of iterations to discard as burn-in.
113             Defaults to 20000.
114
115     Returns:
116         Tuple[torch.Tensor, torch.Tensor, torch.Tensor, float]: Samples of p,
117         w_1, w_2, and the fraction of accepted proposals.
118     """
119     feature_dim = dataset.shape[1]
120
121     # Initialize random starting parameters by sampling priors
122     curr_p = 0.3 + 0.4 * torch.rand(1)
123     curr_w_1 = torch.randn(feature_dim)
124     curr_w_2 = torch.randn(feature_dim)
125
126     # Keep track of samples and total number of accepted proposals
127     p_samples = []
128     w_1_samples = []
129     w_2_samples = []
130     accept_count = 0
131
132     for T in tqdm(range(num_iters)):
133         # YOUR CODE HERE (~3 lines)
134         pass # Sample proposals for p, w_1, w_2
135         # END OF YOUR CODE
136
137         # YOUR CODE HERE (~4-6 lines)
138         pass # Compute likehoods and prior densities on both the proposed and current
    ↪ samples
139         # END OF YOUR CODE
```

```
        # YOUR CODE HERE (~2-4 lines)
        pass # Obtain the ratios of the likelihoods and prior densities between the proposed and current samples
        # END OF YOUR CODE

        # YOUR CODE HERE (~1-2 lines)
        pass # Multiply all ratios (both likelihoods and prior densities) and use this to calculate the acceptance probability of the proposal
        # END OF YOUR CODE

        # YOUR CODE HERE (~4-6 lines)
        pass # Sample randomness to determine whether the proposal should be accepted to update curr_p, curr_w_1, curr_w_2, and accept_count
        # END OF YOUR CODE

        # YOUR CODE HERE (~4-6 lines)
        pass # Update p_samples, w_1_samples, w_2_samples if we have passed the burn in period T
        # END OF YOUR CODE

    fraction_accepted = accept_count / num_iters
    print(f"Fraction of accepted proposals: {fraction_accepted}")
    return (
        torch.stack(p_samples),
        torch.stack(w_1_samples),
        torch.stack(w_2_samples),
        fraction_accepted,
    )


def evaluate_metropolis(num_sims: int, num_movies: int, feature_dim: int) -> None:
    """
    Runs the Metropolis-Hastings algorithm N times and compare estimated parameters with true parameters to obtain success rate. You should attain a success rate of around 90%.

    Note that there are two successful equilibria to converge to. They are true_weights_1 and true_weights_2 with probabilities
    p and 1-p in addition to true_weights_2 and true_weights_1 with probabilities 1-p and p. This is why even though it may appear your
    predicted parameters don't match the true parameters, they are in fact equivalent.

    Args:
        num_sims (int): Number of simulations to run.

    Returns:
        None
```

```
    """

    success_count = 0
    for _ in range(num_sims):
        # Sample random ground truth parameters
        true_p = 0.3 + 0.4 * torch.rand(1)
        true_weights_1 = torch.randn(feature_dim)
        true_weights_2 = torch.randn(feature_dim)

        print("\n---- MCMC Simulation ----")
        print("True parameters:", true_p, true_weights_1, true_weights_2)

        dataset, labels = make_data(true_p, true_weights_1, true_weights_2,
↪   num_movies, feature_dim)
        p_samples, w_1_samples, w_2_samples, _ = metropolis_hastings(dataset, labels)

        p_pred = p_samples.mean(dim=0)
        w_1_pred = w_1_samples.mean(dim=0)
        w_2_pred = w_2_samples.mean(dim=0)

        print("Predicted parameters:", p_pred, w_1_pred, w_2_pred)

        # Do casework on two equilibria cases to check for success
        p_diff_case_1 = torch.abs(p_pred - true_p)
        p_diff_case_2 = torch.abs(p_pred - (1 - true_p))

        w_1_diff_case_1 = torch.max(torch.abs(w_1_pred - true_weights_1))
        w_1_diff_case_2 = torch.max(torch.abs(w_1_pred - true_weights_2))

        w_2_diff_case_1 = torch.max(torch.abs(w_2_pred - true_weights_2))
        w_2_diff_case_2 = torch.max(torch.abs(w_2_pred - true_weights_1))

        pass_case_1 = (
            p_diff_case_1 < 0.1 and w_1_diff_case_1 < 0.5 and w_2_diff_case_1 < 0.5
        )
        pass_case_2 = (
            p_diff_case_2 < 0.1 and w_1_diff_case_2 < 0.5 and w_2_diff_case_2 < 0.5
        )
        passes = pass_case_1 or pass_case_2

        print(f'Result: {"Success" if passes else "FAILED"}')
        if passes:
            success_count += 1
    print(f'Success rate: {success_count / num_sims}')


if __name__ == "__main__":
    evaluate_metropolis(num_sims=10, num_movies=30000, feature_dim=10)
```

References

Andrich, David. 1978. "Relationships Between the Thurstone and Rasch Approaches to Item Scaling." *Applied Psychological Measurement* 2 (3): 451–62.

Armstrong, David A, Ryan Bakker, Royce Carroll, Christopher Hare, Keith T Poole, and Howard Rosenthal. 2014. *Analyzing Spatial Models of Choice and Judgment with r*. CRC Press.

Arrow, Kenneth J. 1951. *Social Choice and Individual Values*. John Wiley; Sons.

Baker, Frank B. 2001. *The Basics of Item Response Theory*. ERIC Clearinghouse on Assessment; Evaluation.

Ben-Akiva, Moshe E, and Steven R Lerman. 1985. *Discrete Choice Analysis: Theory and Application to Travel Demand*. Vol. 9. MIT press.

Bradley, Ralph Allan, and Milton E Terry. 1952. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45.

Breiman, Leo. 2001. "Statistical Modeling: The Two Cultures (with Comments and a Rejoinder by the Author)." *Statistical Science* 16 (3): 199–231. https://doi.org/10.1214/ss/1009213726.

Carroll, J Douglas, and Jih-Jie Chang. 1972. "Individual Differences and Multidimensional Scaling." *Multidimensional Scaling: Theory and Applications in the Behavioral Sciences* 1: 105–55.

Cattelan, Manuela. 2012. "Models for Paired Comparison Data: A Review with Emphasis on Dependent Data." *Statistical Science* 27 (3): 412–33.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. "Deep Reinforcement Learning from Human Preferences." *Advances in Neural Information Processing Systems* 30.

Chu, Wei, and Zoubin Ghahramani. 2005. "Preference Learning with Gaussian Processes." *Proceedings of the 22nd International Conference on Machine Learning*, 137–44.

Coombs, Clyde H. 1950. "Psychological Scaling Without a Unit of Measurement." *Psychological Review* 57 (3): 145–58.

Ganguli, Deep, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, et al. 2022. "Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned." https://arxiv.org/abs/2209.07858.

Gneiting, Tilmann, and Adrian E Raftery. 2007. "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association* 102 (477): 359–78.

He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. "Neural Collaborative Filtering." In *Proceedings of the 26th International Conference on World Wide Web*, 173–82.

Kahneman, Daniel. 2011. *Thinking, Fast and Slow*. Farrar, Straus; Giroux.

Kaufmann, Timo, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. 2023. "A Survey of Reinforcement Learning from Human Feedback." *arXiv Preprint arXiv:2312.14925*.

Koren, Yehuda, Robert Bell, and Chris Volinsky. 2009. "Matrix Factorization Techniques for Recommender Systems." *Computer* 42 (8): 30–37.

Luce, R Duncan et al. 1959. *Individual Choice Behavior*. Vol. 4. Wiley New York.

Mas-Colell, Andreu, Michael Dennis Whinston, Jerry R Green, et al. 1995. *Microeconomic Theory*. Vol. 1. Oxford university press New York.

McFadden, Daniel. 1972. "Conditional Logit Analysis of Qualitative Choice Behavior."

———. 1974. "Conditional Logit Analysis of Qualitative Choice Behavior." *Frontiers in Econometrics*, 105–42.

———. 1981. "Econometric Models of Probabilistic Choice." *Structural Analysis of Discrete Data with Econometric Applications* 198272.

———. 2000. "Disaggregate Behavioral Travel Demand's RUM Side: A 30-Year Retrospective." *Travel Behaviour Research: The Leading Edge*, 17–63.

McFadden, Daniel, and Kenneth Train. 2000. "Mixed MNL Models for Discrete Response." *Journal of Applied Econometrics* 15 (5): 447–70.

Mnih, Andriy, and Ruslan R Salakhutdinov. 2007. "Probabilistic Matrix Factorization." In *Advances in Neural Information Processing Systems*. Vol. 20.

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, et al.

2022. "Training Language Models to Follow Instructions with Human Feedback." *Advances in Neural Information Processing Systems* 35: 27730–44.

Plackett, Robin L. 1975. "The Analysis of Permutations." *Journal of the Royal Statistical Society Series C: Applied Statistics* 24 (2): 193–202.

Poole, Keith T, and Howard Rosenthal. 1985. "A Spatial Model for Legislative Roll Call Analysis." *American Journal of Political Science* 29 (2): 357–84.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." https://arxiv.org/abs/2305.18290.

Rasch, Georg. 1960. *Probabilistic Models for Some Intelligence and Attainment Tests*. Danish Institute for Educational Research.

Rudin, Cynthia, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. 2022. "Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges." *Statistics Surveys* 16: 1–85. https://doi.org/10.1214/21-SS133.

Semenova, Lesia, Cynthia Rudin, and Ronald Parr. 2022. "On the Existence of Simpler Machine Learning Models." *ACM/FAccT*.

Sen, Amartya. 1986. *Social Choice Theory*. Vol. 1. Elsevier.

Siththaranjan, Anand, Cassidy Laidlaw, and Dylan Hadfield-Menell. 2023. "Distributional Preference Learning: Understanding and Accounting for Hidden Context in RLHF." *arXiv Preprint arXiv:2312.08358*.

Skalse, Joar, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. "Defining and Characterizing Reward Hacking." *Advances in Neural Information Processing Systems* 35.

Thurstone, Louis Leon. 1927. "A Law of Comparative Judgment." *Psychological Review* 34 (4): 273–86.

Train, Kenneth E. 2009. *Discrete Choice Methods with Simulation*. Cambridge university press.

Yellott Jr, John I. 1977. "The Relationship Between Luce's Choice Axiom, Thurstone's Theory of Comparative Judgment, and the Double Exponential Distribution." *Journal of Mathematical Psychology* 15 (2): 109–44.

2 Learning

Intended Learning Outcomes

By the end of this chapter you will be able to:

- **Identify** appropriate train/test splitting strategies for preference data and understand their implications for model evaluation.
- **Implement** maximum likelihood estimation via gradient descent for the Bradley-Terry model, deriving and computing gradients efficiently.
- **Apply** Bayesian inference using Markov Chain Monte Carlo (MCMC) to quantify parameter uncertainty and obtain posterior distributions.
- **Apply** Laplace approximation to perform inference for Gaussian Process preference models, understanding why non-Gaussian likelihoods require approximation.
- **Derive** online learning algorithms (Elo ratings) as stochastic gradient ascent and explain when online methods are preferred over batch learning.
- **Compare** batch maximum likelihood, Bayesian inference, and online learning approaches in terms of computational cost, statistical efficiency, and practical applicability.
- **Evaluate** learned preference models using multiple metrics including AUC, log-likelihood, and calibration error.
- **Apply** regularization techniques (L2 penalty, early stopping) to prevent overfitting and improve generalization.
- **Conduct** k-fold cross-validation for model selection and hyperparameter tuning in preference learning settings.
- **Implement** modern optimization methods (Adam, learning rate schedules) and diagnose convergence through loss curves and gradient norms.
- **Analyze** real-world preference data from language model alignment, addressing practical challenges such as noise, sparsity, and cold-start problems.

Suggested Lecture Plan

This chapter can be covered in two 50-minute lectures plus a hands-on lab session:

Lecture 1 (Sections 3.1–3.4): Core parameter estimation methods

- Train/test splitting and evaluation metrics (10 min)
- Maximum likelihood estimation for Bradley-Terry: derivation, implementation, evaluation (20 min)
- Bayesian inference with MCMC: posterior distributions, Metropolis-Hastings (15 min)
- Comparison of MLE vs Bayesian approaches (5 min)

Lecture 2 (Sections 3.5–3.8): Advanced learning topics

- Online learning with Elo ratings: derivation as SGD, applications (15 min)
- Regularization and overfitting: L2 penalty, validation curves (15 min)
- Cross-validation and model selection (10 min)
- Optimization methods: GD vs Adam, learning rate schedules (10 min)

Lab session: Real-world application and exercises

- Apply methods to LLM preference data (30 min)
- Work on selected exercises (20 min)

Designing a good reward signal by hand for a complex AI system is difficult and error-prone. Instead of manually specifying desirable behavior, we can learn a utility signal from preference data. In this chapter, we explore how to infer an underlying utility from various forms of feedback. Throughout, we include mathematical formulations and code examples to illustrate the learning process.

2.1 Chapter Overview

in Chapter 2 established the foundational models for preference data: the Rasch model for item-wise responses, the Bradley-Terry model for pairwise comparisons, and K-dimensional factor models for richer representations. We saw how these models formalize the relationship between latent parameters and observed choices.

This chapter addresses the central question of **learning**: given observed preference data, how do we estimate the parameters of these models? We explore four complementary perspectives on parameter learning, each with distinct advantages:

- **Maximum Likelihood Estimation** (): Find parameters that maximize the probability of observed data. Fast and scalable, but provides only point estimates.
- **Bayesian Inference** (,): Treat parameters as random variables with prior distributions. We cover both parametric models (using MCMC) and nonparametric Gaussian Process models (using Laplace approximation). Quantifies uncertainty but requires more computation.
- **Online Learning** (): Update estimates incrementally as new data arrives. Essential for dynamic systems with continuously arriving feedback.

Beyond these core methods, we cover essential machine learning techniques for practical applications:

- **Regularization** (): Prevent overfitting through L2 penalties and early stopping.
- **Model Selection** (): Use cross-validation to tune hyperparameters and compare approaches.
- **Optimization** (): Improve convergence with modern methods like Adam and adaptive learning rates.
- **Real-World Application** (): Apply all methods to language model preference data, confronting practical challenges.

Chapters 4–6 will build on these learning foundations to address active data collection (Chapter 4), decision-making under learned preferences (Chapter 5), and heterogeneous populations (Chapter 6).

2.2 Learning from Preference Data

introduced the latent variable models for preference data: the Rasch model for item-wise responses, K-dimensional factor models (including the logistic factor model and the ideal point model), and the Bradley-Terry model for pairwise comparisons. We saw how these models relate user parameters U_i and item parameters V_j to observed responses, and how pairwise models can be derived from item-wise models.

This chapter focuses on *learning* the parameters of these models from observed data. We cover three complementary approaches:

1. **Maximum Likelihood Estimation** for Bradley-Terry and Rasch models — finding parameters that maximize the probability of the observed data
2. **Bayesian Inference** for uncertainty quantification — treating parameters as random variables with prior distributions
3. **Online Learning** with the Elo rating system — updating parameter estimates incrementally as new data arrives

We begin by discussing how to split preference data into training and test sets, then develop estimation procedures for each approach.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| autorun: true
#| echo: false
rng = np.random.default_rng(2601)

N = 50 # users
M = 30 # items
U = rng.normal(loc=0.0, scale=1, size=N) # users' appetites
V = rng.normal(loc=0.0, scale=1, size=M) # items' appeals
```

2.3 Maximum Likelihood Estimation

Given a response from a particular generating process, there are various standard statistical inference procedures, such as maximum likelihood, maximum marginal likelihood, or Bayesian inference. These procedures are designed to infer parameters that are generalizable to new datasets. Hence, we discuss the training and testing dataset next.

The simplest train/test splitting is random, where we select a random subset (e.g., 80%) of the response matrix to be in the training set, and the rest in the test set. We can demonstrate, for example, with the Bradley-Terry response with 80% training and 20% testing data:

```
#| autorun: true
V = rng.normal(loc=0.0, scale=1, size=M)
diff = V[:, None] - V[None, :]
P = 1.0 / (1.0 + np.exp(-diff))

Y_BT = np.full((M, M), np.nan)
triu_idx = np.triu_indices(M, k=1)
randu = rng.random(size=triu_idx[0].shape[0])
wins_j_over_k = (randu < P[triu_idx])  # 1 if j beats k
Y_BT[triu_idx] = wins_j_over_k
Y_BT[(triu_idx[1], triu_idx[0])] = 1.0 - wins_j_over_k

# Indices for upper triangle (excluding diagonal)
triu_r, triu_c = np.triu_indices(M, k=1)

# Keep only pairs that are observed (not NaN) in the source matrix
valid_mask = ~np.isnan(Y_BT[triu_r, triu_c])
triu_r = triu_r[valid_mask]
triu_c = triu_c[valid_mask]

n_pairs = triu_r.shape[0]
n_train = int(0.8 * n_pairs)
idx = rng.choice(n_pairs, size=n_train, replace=False)

train_pairs = np.zeros(n_pairs, dtype=bool)
train_pairs[idx] = True
```

```

Y_train = np.full_like(Y_BT, np.nan, dtype=float)
Y_test  = np.full_like(Y_BT, np.nan, dtype=float)

r_tr, c_tr = triu_r[train_pairs], triu_c[train_pairs]
Y_train[r_tr, c_tr] = Y_BT[r_tr, c_tr]
Y_train[c_tr, r_tr] = 1.0 - Y_BT[r_tr, c_tr]

# Fill test set (remaining pairs), both symmetric entries
r_te, c_te = triu_r[~train_pairs], triu_c[~train_pairs]
Y_test[r_te, c_te] = Y_BT[r_te, c_te]
Y_test[c_te, r_te] = 1.0 - Y_BT[r_te, c_te]

# Diagonals stay NaN
np.fill_diagonal(Y_train, np.nan)
np.fill_diagonal(Y_test, np.nan)

# Combined matrix: train (blue/red) and test (green/orange)
# Encode: 0=train-0, 1=train-1, 2=test-0, 3=test-1
Y_combined = np.full((M, M), np.nan)
Y_combined[~np.isnan(Y_train)] = Y_train[~np.isnan(Y_train)]  # 0 or 1 for train
Y_combined[~np.isnan(Y_test)] = Y_test[~np.isnan(Y_test)] + 2  # 2 or 3 for test

from matplotlib.colors import ListedColormap
colors = ['#4575b4', '#d73027', '#1a9850', '#fdae61']  # blue, red, green, orange
cmap = ListedColormap(colors)
cmap.set_bad(color='white')

plt.figure()
plt.imshow(np.ma.masked_invalid(Y_combined), vmin=0, vmax=3, cmap=cmap)
plt.title("Train/Test Split")
plt.xlabel("Items")
plt.ylabel("Items")
import matplotlib.patches as mpatches
legend_patches = [
    mpatches.Patch(color='#4575b4', label='Train: 0'),
    mpatches.Patch(color='#d73027', label='Train: 1'),
    mpatches.Patch(color='#1a9850', label='Test: 0'),
    mpatches.Patch(color='#fdae61', label='Test: 1'),
]
plt.legend(handles=legend_patches, loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```

Having the train and test datasets from the Bradley–Terry response, which are denoted as $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, we first demonstrate the parameter learning with full information maximum likelihood estimation on a single user:

$$\hat{V} = \arg\max_{V} \sum_{j,j' \in \mathcal{D}_{\text{train}}} \log p(Y_{0,jj'} | \sigma(H_{0,jj'})), \quad H_0 = V 1_M^\top - 1_M V^\top, \tag{2.1}$$

where 1_M is a column vector of 1 with length M and $H_{0,jj'} = v_j - v_{j'}$ for a fixed user 0. The optimization can be carried out with standard optimizers, such as gradient descent.

Assumption: Connected Comparison Graph

The MLE for Bradley-Terry parameters exists and is unique if and only if the **comparison graph is strongly connected**—that is, for every pair of items (j, k), there exists a chain of comparisons linking j to k (see Ford Jr 1957). If some items have never been compared (directly or transitively), their utility parameters are not identifiable and the MLE diverges to $\pm\infty$. In practice, sparse datasets (e.g., LLM evaluation where each response pair is annotated only once) frequently produce disconnected comparison graphs. Regularization () addresses this by shrinking all parameters toward zero, effectively imposing a prior that prevents divergence.

Let $\mathcal{N}_m^+ = \{(m, k) \in \mathcal{D}_{\text{train}}\}$ be pairs recorded in the order (m, k). Let $\mathcal{N}_m^- = \{(k, m) \in \mathcal{D}_{\text{train}}\}$ be pairs recorded in the order (k, m). Define residuals $r_{mk} = y_{mk} - \sigma(V_m - V_k)$ and $r_{km} = y_{km} - \sigma(V_k - V_m)$. Then

$$\frac{\partial \ell}{\partial V_m} = \sum_{(m,k) \in \mathcal{N}_m^+} r_{mk} - \sum_{(k,m) \in \mathcal{N}_m^-} r_{km}. \tag{2.2}$$

This shows the contribution of data about m only. Each comparison with opponent k adds a term equal to the observed-minus-predicted win indicator, with a plus sign if m is listed first and a minus sign if m is listed second. Intuitively, if m beats k more often than the model predicts, the residual is positive and the gradient pushes V_m up. If m loses to k more than predicted, the residual is negative and the gradient pushes V_m down.

```
#| autorun: true
def auc_from_scores(scores, labels):
    pos = scores[labels == 1]
    neg = scores[labels == 0]
    if pos.size == 0 or neg.size == 0:
        return np.nan
    # Brute-force definition to avoid dependencies and handle ties
    cmp = (pos[:, None] > neg[None, :]).mean()
    ties = (pos[:, None] == neg[None, :]).mean()
    return cmp + 0.5 * ties

# Collect observed training pairs and labels from the upper triangle only
r_obs, c_obs = np.where(~np.isnan(Y_train))
upper_mask = r_obs < c_obs
r_tr_fit = r_obs[upper_mask]
c_tr_fit = c_obs[upper_mask]
y_tr_fit = Y_train[r_tr_fit, c_tr_fit].astype(float)
```

```

epochs = 100
lr = 0.01
train_auc_hist = []
n_pairs = r_tr_fit.size
V_hat = np.zeros(M, dtype=float)

for t in range(epochs):
    # Scores and probabilities for observed training pairs
    H = V_hat[r_tr_fit] - V_hat[c_tr_fit]
    p = 1.0 / (1.0 + np.exp(-H))
    err = (y_tr_fit - p)
    grad = np.zeros_like(V_hat)
    np.add.at(grad, r_tr_fit, err)
    np.add.at(grad, c_tr_fit, -err)
    V_hat += lr * grad
    train_auc_hist.append(auc_from_scores(H, y_tr_fit))

# Collect observed test pairs and labels from the upper triangle
r_te_obs, c_te_obs = np.where(~np.isnan(Y_test))
upper_mask_te = r_te_obs < c_te_obs
r_te_fit = r_te_obs[upper_mask_te]
c_te_fit = c_te_obs[upper_mask_te]
y_te_fit = Y_test[r_te_fit, c_te_fit].astype(float)

# Scores for test pairs
s_te = V_hat[r_te_fit] - V_hat[c_te_fit]
test_auc = auc_from_scores(s_te, y_te_fit)

# Plot
plt.figure()
plt.plot(np.arange(1, epochs + 1), train_auc_hist, label="Train")
plt.hlines(test_auc, xmin=0, xmax=epochs, linestyle="--", label="Test")
plt.xlabel("Epoch")
plt.ylabel("Train AUC")
plt.ylim(0.75, 0.85)
plt.show()
```

In this synthetic setting, one way to interpret the result is to compare with the Bayes optimal AUC. Here we quantify an upper bound on achievable test AUC under the assumed data-generating process. For each test pair (j, k) we know the ground-truth win probability $P_{jk} = \sigma(V_j - V_k)$ from the simulator. The Bayes-optimal score for that pair is exactly this probability, $s^*_{jk} = P_{jk}$. Any classifier that ranks pairs by s^* maximizes AUC in expectation because it orders pairs by their true success probabilities. To estimate the corresponding Bayes AUC on our finite test set, we keep the same index set of pairs and repeatedly resample binary labels $Y_{jk} \sim \text{Bern}(P_{jk})$, then compute $\text{AUC}(s^*, Y)$ on each resample using a tie-aware definition. The mean of these Monte Carlo AUCs is the Bayes-optimal test AUC, and the empirical quantiles give a sampling range induced purely by label noise. For comparison, we compute the model's

AUC by replacing s^*_{jk} with the learned scores $s^{*\text{model}}_{jk} = \sigma(\hat{V}_j - \hat{V}_k)$. The gap between the two summarizes how far the fitted model is from the oracle ranking implied by the true P^*_{jk} on the exact same test pairs.

```
#| autorun: true
def collect_pairs(Y):
    r, c = np.where(~np.isnan(Y))
    upper = r < c
    return r[upper], c[upper], Y[r[upper], c[upper]].astype(float)

# Unordered test set
r_te_u, c_te_u, y_te_u = collect_pairs(Y_test)

# Bayes scores and model scores for those exact pairs
s_bayes = P[r_te_u, c_te_u]                  # ground-truth probs
s_model = 1.0/(1.0+np.exp(-(V_hat[r_te_u]-V_hat[c_te_u])))

auc_bayes = auc_from_scores(s_bayes, y_te_u)
auc_model = auc_from_scores(s_model, y_te_u)
print(f"Test AUC. Bayes: {auc_bayes:.4f}. Model: {auc_model:.4f}.")

def resample_labels_from_P(r_idx, c_idx, P_mat, rng):
    p = P_mat[r_idx, c_idx]
    return (rng.random(size=p.size) < p).astype(float)

R = 2000  # repeats
bayes_list = []
model_list = []
for _ in range(R):
    y_resamp = resample_labels_from_P(r_te_u, c_te_u, P, rng)
    bayes_list.append(auc_from_scores(s_bayes, y_resamp))
    model_list.append(auc_from_scores(s_model, y_resamp))

print(f"Mean AUC over resamples. Bayes: {np.mean(bayes_list):.4f}. Model:
 ↪  {np.mean(model_list):.4f}.")
print(f"5-95% range Bayes: {np.percentile(bayes_list,[5,95])}. Model:
 ↪  {np.percentile(model_list,[5,95])}.")
```

The result shows that the inference has found a good solution compared to the Bayes optimal estimator! An additional, natural test is to see if the estimated parameters match the true ones. Since the likelihood function only pays attention to the difference of item appeal for any pair, the appeal is only identifiable up to a shift and scale transform. The standard practice is to center and whiten the solution.

```
#| autorun: true
# Center and whiten V_hat (zero mean, unit variance)
V_hat_norm = (V_hat - V_hat.mean()) / (V_hat.std() + 1e-12)
```

```
# Fit best affine map a*V_hat_norm + b to V (least squares)
A = np.vstack([V_hat_norm, np.ones_like(V_hat_norm)]).T
a, b = np.linalg.lstsq(A, V, rcond=None)[0]
V_hat_aligned = a * V_hat_norm + b

# After alignment, points should cluster near the diagonal
plt.figure()
plt.scatter(V, V_hat, s=12, c="red")
plt.scatter(V, V_hat_aligned, s=12, c="blue")
lims = [-3, 3]
plt.plot(lims, lims, linestyle="--")
plt.xlabel("V")
plt.ylabel("Estimated V")

# Report alignment quality
corr = np.corrcoef(V, V_hat)[0, 1]
mse = np.mean((V - V_hat) ** 2)
print(f"Before center and whiten: Corr {corr:.4f}, MSE: {mse:.6f}")

corr = np.corrcoef(V, V_hat_aligned)[0, 1]
mse = np.mean((V - V_hat_aligned) ** 2)
print(f"After center and whiten: Corr {corr:.4f}, MSE: {mse:.6f}")
```

Looking Ahead: Regularization

MLE can overfit when parameters outnumber observations. introduces L2 regularization, which adds a penalty $\frac{\lambda}{2}\|V\|_2^2$ to the objective. This is equivalent to MAP estimation with a Gaussian prior—connecting MLE to the Bayesian framework we develop next.

2.4 Bayesian Inference

A Bayesian approach provides a natural alternative to maximum likelihood for parameter estimation. Instead of finding a single point estimate, we place a prior distribution on parameters and update it using the likelihood from observed comparisons. This yields a posterior distribution that captures both central estimates and the uncertainty around them. We first cover parametric models using MCMC, then extend to nonparametric Gaussian Process models using Laplace approximation.

2.4.1 Parametric Models

For the Bradley–Terry model with finite-dimensional item parameters V, we place i.i.d. standard normal priors and update them using the Bernoulli likelihood from observed comparisons. In practice, this posterior cannot be computed in closed form, so we turn to Markov chain

Monte Carlo (MCMC), which constructs a sequence of samples that, in the limit, follow the posterior distribution.

The Metropolis–Hastings (MH) algorithm is straightforward to apply: at each step, we propose a new value for one or more coordinates of (V) (e.g., from a Gaussian centered at the current state), compute the acceptance ratio as the ratio of posterior densities between the proposed and current states, and accept the proposal with that probability. Repeating this process produces a chain of samples that can be used to approximate posterior means, variances, or other functionals of interest. This approach not only yields point predictions but also quantifies uncertainty about the relative strengths of items under the Bradley–Terry model.

Let the prior be i.i.d. standard normal for each item parameter: $p(V) = \prod_{j=1}^{M} \mathcal{N}(V_j \mid 0, 1)$. Then the posterior distribution is $p(V \mid \mathcal{D}) = p(\mathcal{D} \mid V)p(V)/p(\mathcal{D})$, where the likelihood is

$$p(\mathcal{D} \mid V) = \prod_{(j,j') \in \mathcal{J}} \sigma(V_j - V_{j'})^{Y_{jj'}} \left(1 - \sigma(V_j - V_{j'})\right)^{1 - Y_{jj'}}. \tag{2.3}$$

The denominator is the marginal likelihood (evidence), which is intractable in closed form, so we resort to MCMC (e.g., MH) to sample from the posterior. Suppose we are at current (parameter) state $V^{(t)}$. We propose a new state V' from a proposal distribution, such as Gaussian: $q(V' \mid V^{(t)}) = \mathcal{N}(V'; V^{(t)}, \tau^2 I)$, where τ^2 is a step-size variance. This proposal is symmetric, meaning $q(V' \mid V^{(t)}) = q(V^{(t)} \mid V')$. For any proposal distribution $q(V' \mid V^{(t)})$, the Metropolis–Hastings acceptance probability is

$$\alpha = \min \left\{ 1, \frac{p(V' \mid \mathcal{D}) \cdot q(V^{(t)} \mid V')}{p(V^{(t)} \mid \mathcal{D}) \cdot q(V' \mid V^{(t)})} \right\}. \tag{2.4}$$

This says: accept the proposal with probability proportional to how much more plausible it is under the posterior, adjusted by how easy it is to propose back. If we choose a Gaussian proposal, the proposal terms cancel out in the ratio due to symmetry. So the acceptance rule simplifies to

$$\alpha = \min \left\{ 1, \frac{p(V' \mid \mathcal{D})}{p(V^{(t)} \mid \mathcal{D})} \right\} = \min \left\{ 1, \frac{p(\mathcal{D} \mid V') \cdot p(V')}{p(\mathcal{D} \mid V^{(t)}) \cdot p(V^{(t)})} \right\}. \tag{2.5}$$

```
#| autorun: true
def logpost(v, r_idx, c_idx, y_obs, prior_var=1.0):
    s = v[r_idx] - v[c_idx]
    # Stable log-sigmoid pieces
    # log p = -softplus(-s), log(1-p) = -softplus(s)
    ll = (y_obs * -np.log1p(np.exp(-s)) + (1 - y_obs) * -np.log1p(np.exp(s))).sum()
    lp = -0.5 * np.dot(v, v) / prior_var
    return ll + lp

```

```
steps = 10000
prop_scale = 0.05

# Test labels (upper)
y_te = Y_test[r_te, c_te].astype(float)

rng = np.random.default_rng(3)
v_cur = rng.normal(scale=0.1, size=M)
v_cur -= v_cur.mean()
lp_cur = logpost(v_cur, r_tr_fit, c_tr_fit, y_tr_fit)

trace_every = 1
trace = []
acc = 0

for t in range(steps):
    v_prop = v_cur.copy()
    # Single-coordinate random walk
    j = rng.integers(M)
    v_prop[j] += rng.normal(scale=prop_scale)
    # Fix shift invariance
    v_prop -= v_prop.mean()
    lp_prop = logpost(v_prop, r_tr_fit, c_tr_fit, y_tr_fit)
    if np.log(rng.random()) < (lp_prop - lp_cur):
        v_cur, lp_cur = v_prop, lp_prop
        acc += 1
    if (t + 1) % trace_every == 0:
        trace.append(v_cur.copy())

trace = np.array(trace)  # shape (steps/trace_every, M)
acc_rate = acc / steps
print(f"[MH] Acceptance rate: {acc_rate:.3f} with prop_scale={prop_scale}")
```

Assumption: MCMC Has Converged

Metropolis-Hastings produces samples from the posterior *asymptotically*, but in finite time, the chain may not have converged. A reasonable acceptance rate (20–50%) is necessary but not sufficient: the chain may mix slowly between modes or explore only a local region of the posterior. In practice, one should run **multiple independent chains** from dispersed starting points and check convergence using diagnostics such as the Gelman-Rubin $\hat{R}$ statistic (which should be close to 1.0) and effective sample size. The Laplace approximation introduced below avoids sampling entirely but assumes the posterior is unimodal and approximately Gaussian—an assumption that fails when data is sparse or the posterior is multimodal.

```
#| autorun: true
# Show trace with posterior marginal
plt.figure()
```

```
fig, (ax_trace, ax_hist) = plt.subplots(1, 2, width_ratios=[4, 1],
                                        sharey=True, layout='constrained')
fig.get_layout_engine().set(wspace=0.05)

# Trace plot
ax_trace.plot(trace[:, 0], lw=0.8, color='steelblue')
ax_trace.set_xlabel("Iteration / 5")
ax_trace.set_ylabel(r"$v_0$")
ax_trace.set_title("Trace of $v_0$")

# Posterior marginal (horizontal histogram)
ax_hist.hist(trace[:, 0], bins=50, density=True, orientation='horizontal',
             color='steelblue', alpha=0.7, edgecolor='white', linewidth=0.5)
ax_hist.set_xlabel("Density")
ax_hist.tick_params(labelleft=False)
ax_hist.set_title("Posterior")

plt.show()
```

2.4.2 Gaussian Processes

MCMC works well for finite-dimensional parameter vectors, but what if we want to learn a *nonparametric* reward function? Gaussian Processes (GPs) extend Bayesian inference to *function spaces*, placing a prior distribution over reward functions $r \sim \mathcal{GP}(m, k)$ and combining it with the Bradley-Terry likelihood for preferences. The challenge is that this likelihood is *non-Gaussian*, breaking the standard GP posterior formulas and requiring approximations like Laplace.

The inference problem. Given pairwise comparisons $\mathcal{D} = \{(x_A^{(i)}, x_B^{(i)}, y_i)\}_{i=1}^n$ where $y_i = 1$ if item A was preferred, we want to compute the posterior:

$$p(r \mid \mathcal{D}) \propto p(\mathcal{D} \mid r) \cdot p(r) \tag{2.6}$$

The likelihood follows the Bradley-Terry model:

$$p(y_i = 1 \mid r) = \sigma\left(r(x_A^{(i)}) - r(x_B^{(i)})\right) \tag{2.7}$$

Because this sigmoid likelihood is not Gaussian, the posterior $p(r \mid \mathcal{D})$ is no longer a Gaussian Process.

2.4.2.1 Laplace Approximation

The **Laplace approximation** provides a tractable Gaussian approximation to the true posterior:

1. **Find the posterior mode** $\mathbf{r}^* = \arg\max_{\mathbf{r}} \log p(\mathcal{D} \mid \mathbf{r}) + \log p(\mathbf{r})$
2. **Approximate with a Gaussian** centered at the mode, using the Hessian of the log-posterior as precision

For preference data with GP prior (zero mean), the mode satisfies:

$$\mathbf{r}^* = \arg\max_{\mathbf{r}} \sum_{i=1}^{n} \log \sigma\left(y_i(r(x_A^{(i)}) - r(x_B^{(i)}))\right) - \frac{1}{2}\mathbf{r}^\top K^{-1}\mathbf{r} \tag{2.8}$$

where K is the kernel matrix and $\mathbf{r} = [r(x_1), \ldots, r(x_m)]^\top$ collects function values at all unique points appearing in comparisons.

Newton's method for finding the mode. The gradient and Hessian of the log-posterior are:

$$\nabla_{\mathbf{r}} \log p(\mathbf{r} \mid \mathcal{D}) = \mathbf{g} - K^{-1}\mathbf{r} \tag{2.9}$$

$$\nabla^2_{\mathbf{r}} \log p(\mathbf{r} \mid \mathcal{D}) = -W - K^{-1} \tag{2.10}$$

where $\mathbf{g}$ is the gradient of the log-likelihood and W is a diagonal matrix with entries $W_{ii} = p_i(1 - p_i)$ for predictions $p_i = \sigma(r(x_A^{(i)}) - r(x_B^{(i)}))$.

The approximate posterior. After finding $\mathbf{r}^*$, the Laplace approximation gives:

$$p(\mathbf{r} \mid \mathcal{D}) \approx \mathcal{N}\left(\mathbf{r}^*, (K^{-1} + W)^{-1}\right) \tag{2.11}$$

This is structurally similar to standard GP regression, but with the data-dependent precision matrix W arising from the Bradley-Terry likelihood rather than observation noise.

Limitations of the Laplace Approximation

The Laplace approximation replaces the true posterior with a Gaussian centered at the MAP estimate $\mathbf{r}^*$. This approximation can be poor when: (1) the posterior is **multimodal** (e.g., when items form disconnected clusters with few cross-cluster comparisons), (2) the MAP is near a **boundary** of the parameter space, or (3) there are **very few comparisons** so the likelihood barely constrains the posterior. In these settings, the Gaussian approximation underestimates posterior uncertainty and can produce overconfident predictions. Alternatives include expectation propagation (EP), which approximates each likelihood factor separately, or full MCMC sampling as described above.

```
#| autorun: true
def rbf_kernel(X1, X2, sigma_f=1.0, length_scale=0.5):
    """RBF kernel matrix."""
    sq_dist = np.sum(X1**2, axis=1, keepdims=True) + np.sum(X2**2, axis=1) - 2 * X1 @
    ↪ X2.T
    return sigma_f**2 * np.exp(-sq_dist / (2 * length_scale**2))

```

```
sigmoid = lambda x: 1.0 / (1.0 + np.exp(-np.clip(x, -500, 500)))

# True reward function (nonlinear!)
def true_reward(x):
    return np.sin(2 * x) + 0.5 * x

# Generate synthetic preference data
n_comparisons = 30
X_A = np.random.uniform(-3, 3, n_comparisons).reshape(-1, 1)
X_B = np.random.uniform(-3, 3, n_comparisons).reshape(-1, 1)
r_A = true_reward(X_A.flatten())
r_B = true_reward(X_B.flatten())
y = (np.random.rand(n_comparisons) < sigmoid(r_A - r_B)).astype(float)

# Collect unique points
X_all = np.vstack([X_A, X_B])
X_unique, inv_idx = np.unique(X_all.flatten(), return_inverse=True)
X_unique = X_unique.reshape(-1, 1)
m = len(X_unique)
idx_A = inv_idx[:n_comparisons]
idx_B = inv_idx[n_comparisons:]

# Build kernel matrix
K = rbf_kernel(X_unique, X_unique) + 1e-4 * np.eye(m)
K_inv = np.linalg.inv(K)

# Laplace approximation via Newton's method
r = np.zeros(m)
for iteration in range(20):
    # Compute predictions and likelihood gradient
    diff = r[idx_A] - r[idx_B]
    p = sigmoid(diff)

    # Gradient of log-likelihood w.r.t. r at each unique point
    grad = np.zeros(m)
    for i in range(n_comparisons):
        residual = y[i] - p[i]
        grad[idx_A[i]] += residual
        grad[idx_B[i]] -= residual

    # Full gradient including GP prior
    grad_total = grad - K_inv @ r

    # Hessian: diagonal from likelihood + K_inv from prior
    W = np.zeros(m)
    for i in range(n_comparisons):
        W[idx_A[i]] += p[i] * (1 - p[i])
        W[idx_B[i]] += p[i] * (1 - p[i])

```

```
    H = -np.diag(W) - K_inv

    # Newton step
    r = r - np.linalg.solve(H, grad_total)

# Posterior covariance at mode
Sigma_post = np.linalg.inv(K_inv + np.diag(W))

# Predictions on a grid
X_grid = np.linspace(-3, 3, 100).reshape(-1, 1)
K_star = rbf_kernel(X_grid, X_unique)
mu_pred = K_star @ K_inv @ r
K_star_star = rbf_kernel(X_grid, X_grid)
var_pred = np.diag(K_star_star - K_star @ (K_inv - K_inv @ Sigma_post @ K_inv) @
 ↪  K_star.T)
std_pred = np.sqrt(np.maximum(var_pred, 0))

# Plot
plt.figure()
plt.fill_between(X_grid.flatten(), mu_pred - 2*std_pred, mu_pred + 2*std_pred,
                 alpha=0.3, color='blue', label='95% CI')
plt.plot(X_grid, mu_pred, 'b-', lw=2, label='GP posterior mean')
plt.plot(X_grid, true_reward(X_grid), 'k--', lw=2, label='True reward')
plt.scatter(X_A[y==1], np.zeros(int(y.sum())) - 2.5, c='green', marker='^', s=50,
 ↪  label='Preferred A')
plt.scatter(X_B[y==0], np.zeros(int((1-y).sum())) - 2.5, c='red', marker='v', s=50,
 ↪  label='Preferred B')
plt.xlabel('x')
plt.ylabel('r(x)')
plt.title(f'GP Preference Learning via Laplace Approximation ({n_comparisons}
 ↪  comparisons)')
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"Learned from {n_comparisons} pairwise comparisons")
print(f"Posterior uncertainty varies: min std = {std_pred.min():.3f}, max std =
 ↪  {std_pred.max():.3f}")
```

2.4.2.2 Connection to Fisher Information

The Laplace approximation has a natural connection to the Fisher information framework we developed for linear models. The Hessian of the negative log-likelihood at the mode gives the *observed Fisher information*:

$$I_{\text{obs}} = W = \text{diag}(p_1(1-p_1), \dots, p_n(1-p_n)) \tag{2.12}$$

Just as with linear preference models, comparisons with $p \approx 0.5$ contribute the most information (highest $p(1-p)$), while "easy" comparisons where one option clearly dominates contribute little.

2.4.2.3 Computational Considerations

Complexity. Each Newton iteration requires $O(m^3)$ computation for the matrix solve, where m is the number of unique points. For n comparisons involving $m \leq 2n$ unique points, the total cost is $O(n^3)$ per iteration.

Scalability. For large datasets, several approximations are available: - **Inducing points**: Approximate the GP using $k \ll m$ pseudo-inputs, reducing complexity to $O(k^2 n)$ - **Variational inference**: Optimize a lower bound on the marginal likelihood - **Conjugate gradient methods**: Avoid explicit matrix inversion

Hyperparameter learning. The kernel hyperparameters (length-scale ℓ, signal variance σ_f^2) can be learned by maximizing the Laplace-approximated marginal likelihood:

$$\log p(\mathcal{D} \mid \theta) \approx \log p(\mathcal{D} \mid \mathbf{r}^*) + \log p(\mathbf{r}^* \mid \theta) + \frac{1}{2} \log |K^{-1} + W|^{-1} \tag{2.13}$$

In Chapter 4, we will see how the GP posterior uncertainty enables *active* query selection—choosing which comparisons to ask to learn most efficiently.

2.5 Online Learning

In many applications, comparisons between items arrive sequentially over time rather than being observed all at once. For example, players in online games are continuously matched, or recommendation systems log one user preference at a time. In such settings, it is often computationally infeasible to refit the full Bradley–Terry model by maximum likelihood or MCMC after each new observation. Instead, we want an online update rule that adjusts item strengths incrementally as new outcomes arrive.

This is precisely the motivation behind the **Elo rating system**, originally introduced for ranking chess players and later widely adopted in competitive games, online platforms, and even information retrieval. The key idea is to maintain a current estimate of each item's (or player's) latent strength, and update only the two items involved in a match when a new result comes in.

The Elo rule can be derived as a stochastic gradient ascent method on the Bradley–Terry log-likelihood. Suppose item j plays against item j'. Given the current parameters, the log-likelihood gradient with respect to V_j and $V_{j'}$ is

$$\frac{\partial \ell}{\partial V_j} = (y - p), \qquad \frac{\partial \ell}{\partial V_{j'}} = -(y - p). \tag{2.14}$$

A stochastic gradient ascent step with learning rate η gives the update:

$$V_j \leftarrow V_j + \eta(y - p), \qquad V_{j'} \leftarrow V_{j'} - \eta(y - p). \tag{2.15}$$

This is exactly the Elo update rule. The learning rate η is often called the **K-factor** in Elo literature. If $y = 1$ (item j wins) but the model predicted a low p, then $(y - p)$ is positive and V_j increases, $V_{j'}$ decreases — the system learns that j is stronger than previously believed. If $y = 0$ (item j' wins), the opposite adjustment happens. The magnitude of the update is larger when the outcome is surprising (large prediction error), and smaller when the outcome is expected (small prediction error). Thus, Elo is an online learning algorithm for the Bradley–Terry model, interpretable as stochastic gradient ascent with a fixed step size.

Worked Example: Elo as Stochastic Gradient Ascent

Two chess players have current ratings $V_A = 1500$ and $V_B = 1700$ (using the traditional Elo scale where σ operates on $(V_A - V_B)/400$). With $K = 32$:

Step 1. Predicted win probability for A:

$$p = \sigma\left(\frac{V_A - V_B}{400}\right) = \sigma\left(\frac{-200}{400}\right) = \sigma(-0.5) \approx 0.378$$

Player A is expected to win about 38% of the time.

Step 2. Suppose A wins (upset!). The prediction error is $y - p = 1 - 0.378 = 0.622$. Updates:

$$V_A \leftarrow 1500 + 32 \times 0.622 = 1500 + 19.9 \approx 1520$$

$$V_B \leftarrow 1700 - 32 \times 0.622 = 1700 - 19.9 \approx 1680$$

Step 3. If instead the favorite B had won ($y = 0$, prediction error $= -0.378$):

$$V_A \leftarrow 1500 - 32 \times 0.378 \approx 1488$$

$$V_B \leftarrow 1700 + 32 \times 0.378 \approx 1712$$

Key insight: Upsets produce larger rating changes (19.9 points) than expected outcomes (12.1 points). This is exactly stochastic gradient ascent: the step size is proportional to the prediction error, so surprising results move parameters more. The K-factor controls the tradeoff between responsiveness (large K) and stability (small K).

Limitations of Fixed K-Factor Elo

The Elo system with a constant K-factor has several limitations. First, it weights all comparisons equally regardless of recency—there is no discounting of old information, so it cannot optimally track **non-stationary** preferences (e.g., an LLM improving through training). Second, a single K must trade off responsiveness against stability for *all* items, even though some items' utilities may be well-established while others are new. Third, unlike Bayesian methods, Elo provides **no uncertainty quantification**—the rating V_j is a point estimate with no associated confidence interval. Systems like Glicko address some of these issues by maintaining per-item

uncertainty estimates that decay with inactivity and scale the update magnitude accordingly.

```
#| autorun: true
# Build a stream of observed pairs for the single user from Y_BT upper triangle
pairs_stream = list(zip(triu_r, triu_c))
labels_stream = Y_BT[triu_r, triu_c].astype(float)

# Initialize ratings r (Elo analog of v)
r = np.zeros(M)
Kfactor = 1.0  # step size

def logistic_prob(a, b):
    return 1.0 / (1.0 + np.exp(-(a - b)))

for (j, k), y in zip(pairs_stream, labels_stream):
    p = logistic_prob(r[j], r[k])
    # Update: winner gets +K*(1-p), loser gets -K*(1-p)
    # If y=1 means j wins. If y=0 means k wins.
    if y == 1.0:
        r[j] += Kfactor * (1 - p)
        r[k] -= Kfactor * (1 - p)
    else:
        r[k] += Kfactor * p
        r[j] -= Kfactor * p
    # optional centering
    r -= r.mean()

# Evaluate Elo ratings on held-out test pairs (same y_te from above)
p_te_elo = 1.0 / (1.0 + np.exp(-(r[r_te] - r[c_te])))
auc_elo = auc_from_scores(p_te_elo, y_te)
```

2.6 Regularization and Overfitting

The maximum likelihood estimator maximizes fit to observed training data but may overfit, especially when the number of parameters is large relative to the amount of data. In preference learning, overfitting manifests as learned item parameters that capture noise in the training comparisons rather than true relative strengths. Regularization techniques penalize model complexity to improve generalization to held-out test data.

2.6.1 L2 Regularization

The most common regularization approach adds an L2 penalty term to the log-likelihood. For the Bradley-Terry model, the regularized objective becomes:

$$\mathcal{L}_{\text{reg}}(V) = \sum_{(j,j') \in \mathcal{D}_{\text{train}}} \log p(Y_{jj'} \mid V_j - V_{j'}) - \frac{\lambda}{2} \|V\|_2^2 \tag{2.16}$$

where $\lambda \geq 0$ is the **regularization strength**. The penalty $\frac{\lambda}{2}\|V\|_2^2 = \frac{\lambda}{2}\sum_{j=1}^{M} V_j^2$ discourages large parameter values. When $\lambda = 0$, we recover standard MLE. As λ increases, parameters shrink toward zero.

The regularized gradient adds a simple term to the MLE gradient:

$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial V_m} = \sum_{(m,k) \in \mathcal{N}_m^+} r_{mk} - \sum_{(k,m) \in \mathcal{N}_m^-} r_{km} - \lambda V_m \tag{2.17}$$

Intuitively, regularization creates a bias-variance tradeoff: small λ permits complex fits (low bias, high variance), while large λ enforces simpler models (high bias, low variance). The optimal λ balances these to minimize test error.

> **CONNECTION TO BAYESIAN INFERENCE**
>
> L2 regularization corresponds exactly to maximum a posteriori (MAP) estimation with a Gaussian prior $p(V_j) = \mathcal{N}(0, 1/\lambda)$. The regularization strength λ is the inverse prior variance: strong regularization (large λ) means a tight prior belief that parameters are near zero.

```
#| autorun: true
# Compare unregularized vs regularized Bradley-Terry on synthetic data
# Generate data with few items and limited comparisons to induce overfitting

rng_reg = np.random.default_rng(42)
M_small = 10  # few items
V_true_small = rng_reg.normal(0, 1, M_small)

# Generate limited training data (sparse comparisons)
n_train_pairs = 30  # only 30 comparisons among 10 items
train_pairs_idx = rng_reg.choice(M_small * (M_small - 1) // 2, n_train_pairs,
 ↪  replace=False)
all_pairs = [(i, j) for i in range(M_small) for j in range(i+1, M_small)]
train_pairs = [all_pairs[idx] for idx in train_pairs_idx]

# Generate outcomes
Y_train_sparse = np.full((M_small, M_small), np.nan)
for (i, j) in train_pairs:
```

```
    p_win = 1.0 / (1.0 + np.exp(-(V_true_small[i] - V_true_small[j])))
    outcome = 1.0 if rng_reg.random() < p_win else 0.0
    Y_train_sparse[i, j] = outcome
    Y_train_sparse[j, i] = 1.0 - outcome

# Generate test data (all remaining pairs)
test_pairs = [pair for pair in all_pairs if pair not in train_pairs]
Y_test_sparse = np.full((M_small, M_small), np.nan)
for (i, j) in test_pairs:
    p_win = 1.0 / (1.0 + np.exp(-(V_true_small[i] - V_true_small[j])))
    outcome = 1.0 if rng_reg.random() < p_win else 0.0
    Y_test_sparse[i, j] = outcome
    Y_test_sparse[j, i] = 1.0 - outcome

# Extract training and test pair indices
r_tr_s, c_tr_s, y_tr_s = collect_pairs(Y_train_sparse)
r_te_s, c_te_s, y_te_s = collect_pairs(Y_test_sparse)

def fit_regularized_bt(r_idx, c_idx, y_obs, M, lam=0.0, lr=0.05, epochs=200):
    """Fit Bradley-Terry with L2 regularization."""
    V = np.zeros(M)
    for _ in range(epochs):
        H = V[r_idx] - V[c_idx]
        p = 1.0 / (1.0 + np.exp(-H))
        err = y_obs - p
        grad = np.zeros(M)
        np.add.at(grad, r_idx, err)
        np.add.at(grad, c_idx, -err)
        # Add regularization gradient
        grad -= lam * V
        V += lr * grad
    return V

# Fit models with different regularization strengths
lambdas = [0.0, 0.01, 0.1, 0.5, 1.0, 5.0]
train_aucs = []
test_aucs = []

for lam in lambdas:
    V_fit = fit_regularized_bt(r_tr_s, c_tr_s, y_tr_s, M_small, lam=lam)
    # Train AUC
    s_tr = V_fit[r_tr_s] - V_fit[c_tr_s]
    train_aucs.append(auc_from_scores(s_tr, y_tr_s))
    # Test AUC
    s_te = V_fit[r_te_s] - V_fit[c_te_s]
    test_aucs.append(auc_from_scores(s_te, y_te_s))

# Plot validation curve
plt.figure()
```

```
plt.semilogx(lambdas, train_aucs, 'o-', label='Train AUC', markersize=8)
plt.semilogx(lambdas, test_aucs, 's-', label='Test AUC', markersize=8)
plt.xlabel(r'$\lambda$')
plt.ylabel('AUC')
plt.title('Validation Curve')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Report optimal lambda
best_idx = np.argmax(test_aucs)
print(f"Optimal  : {lambdas[best_idx]:.3f} (Test AUC: {test_aucs[best_idx]:.4f})")
print(f"Unregularized ( =0): Train AUC {train_aucs[0]:.4f}, Test AUC
 ↪  {test_aucs[0]:.4f}")
```

The validation curve demonstrates the regularization trade-off: at $\lambda = 0$ (no regularization), the model overfits to training data, achieving high training AUC but lower test AUC. As λ increases, test performance improves until an optimal point, after which excessive regularization underfits. The gap between training and test AUC narrows with proper regularization, indicating better generalization.

2.6.2 Early Stopping

An alternative to explicit regularization is **early stopping**: monitor validation performance during training and stop when it begins to degrade, even if training performance continues improving. This exploits the empirical observation that gradient descent follows a path from simple to complex models.

```
#| autorun: true
# Demonstrate early stopping
def fit_bt_with_validation(r_tr, c_tr, y_tr, r_val, c_val, y_val, M, lr=0.01,
 ↪  max_epochs=300):
    """Fit Bradley-Terry with early stopping."""
    V = np.zeros(M)
    train_auc_hist = []
    val_auc_hist = []

    for epoch in range(max_epochs):
        # Gradient step
        H = V[r_tr] - V[c_tr]
        p = 1.0 / (1.0 + np.exp(-H))
        err = y_tr - p
        grad = np.zeros(M)
        np.add.at(grad, r_tr, err)
        np.add.at(grad, c_tr, -err)
        V += lr * grad
```

```
        # Evaluate
        s_tr = V[r_tr] - V[c_tr]
        train_auc_hist.append(auc_from_scores(s_tr, y_tr))
        s_val = V[r_val] - V[c_val]
        val_auc_hist.append(auc_from_scores(s_val, y_val))

    return V, train_auc_hist, val_auc_hist

V_es, tr_hist, val_hist = fit_bt_with_validation(
    r_tr_s, c_tr_s, y_tr_s, r_te_s, c_te_s, y_te_s, M_small
)

# Find early stopping point (best validation performance)
best_epoch = np.argmax(val_hist)

plt.figure()
plt.plot(tr_hist, label='Train AUC', alpha=0.7)
plt.plot(val_hist, label='Validation AUC', alpha=0.7)
plt.axvline(best_epoch, color='red', linestyle='--', label=f'Best epoch:
 ↪  {best_epoch}')
plt.xlabel('Epoch')
plt.ylabel('AUC')
plt.title('Early Stopping: Training vs Validation Performance')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"Best validation AUC {val_hist[best_epoch]:.4f} at epoch {best_epoch}")
print(f"Final epoch ({len(val_hist)-1}): Train {tr_hist[-1]:.4f}, Val
 ↪  {val_hist[-1]:.4f}")
```

Early stopping automatically selects model complexity through the training trajectory. Unlike L2 regularization, it requires no tuning of λ, but does require held-out validation data to monitor.

! Key Takeaway: When to Regularize

Regularization is most critical when:

- **Few observations** relative to parameters (e.g., 50 comparisons, 20 items)
- **Imbalanced data**: Some items have many comparisons, others have few
- **High noise**: Label noise or measurement error in comparisons

For large datasets with many comparisons per item, overfitting is less of a concern and regularization may have minimal impact. Always use validation data or cross-validation to tune regularization strength.

2.7 Model Selection and Cross-Validation

A single train/test split provides one estimate of generalization performance, but this estimate has high variance: a different random split may yield different results. **Cross-validation** (CV) systematically evaluates multiple train/test splits to obtain more reliable performance estimates and enable principled model selection.

2.7.1 K-Fold Cross-Validation

In **k-fold cross-validation**, we partition the data into k equally-sized folds. For each fold $i \in \{1, \ldots, k\}$:

1. Train the model on all folds except fold i
2. Evaluate on fold i (held-out validation)
3. Record the validation metric (e.g., AUC, log-likelihood)

The final CV score is the average across all k folds: $\text{CV}_k = \frac{1}{k} \sum_{i=1}^{k} \text{metric}_i$. The standard error quantifies uncertainty in the estimate.

For preference data, we partition the set of observed pairwise comparisons (not items) into folds, ensuring each fold contains a representative sample of comparisons.

```
#| autorun: true
def kfold_split_pairs(r_idx, c_idx, y_obs, k=5, rng=None):
    """Split pairwise comparison data into k folds."""
    if rng is None:
        rng = np.random.default_rng()
    n_pairs = len(r_idx)
    fold_ids = np.arange(n_pairs) % k
    rng.shuffle(fold_ids)
    return fold_ids

def cross_validate_bt(r_idx, c_idx, y_obs, M, k=5, lam=0.0, lr=0.05, epochs=200):
    """Perform k-fold cross-validation for Bradley-Terry model."""
    rng_cv = np.random.default_rng(123)
    fold_ids = kfold_split_pairs(r_idx, c_idx, y_obs, k, rng_cv)

    fold_scores = []
    for fold in range(k):
        # Split into train and validation
        train_mask = fold_ids != fold
        val_mask = fold_ids == fold

        r_tr = r_idx[train_mask]
        c_tr = c_idx[train_mask]
        y_tr = y_obs[train_mask]

        r_val = r_idx[val_mask]
```

```
        c_val = c_idx[val_mask]
        y_val = y_obs[val_mask]

        # Train model
        V = fit_regularized_bt(r_tr, c_tr, y_tr, M, lam=lam, lr=lr, epochs=epochs)

        # Evaluate on validation fold
        s_val = V[r_val] - V[c_val]
        auc_val = auc_from_scores(s_val, y_val)
        fold_scores.append(auc_val)

    return np.array(fold_scores)

# Perform 5-fold CV on the synthetic dataset
cv_scores = cross_validate_bt(r_tr_s, c_tr_s, y_tr_s, M_small, k=5, lam=0.1)

print(f"5-Fold CV Scores: {cv_scores}")
print(f"Mean CV AUC: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")
```

The CV estimate of {cv_scores.mean():.4f} ± {cv_scores.std():.4f} is more reliable than a single train/test split. The standard deviation quantifies variability across folds.

2.7.2 Hyperparameter Tuning with Cross-Validation

Cross-validation enables principled hyperparameter selection: evaluate each candidate hyperparameter on CV performance and select the best. For Bradley-Terry, key hyperparameters include regularization strength λ and learning rate.

```
#| autorun: true
# Grid search over regularization strengths using cross-validation
lambda_grid = [0.0, 0.01, 0.05, 0.1, 0.5, 1.0]
cv_means = []
cv_stds = []

for lam in lambda_grid:
    scores = cross_validate_bt(r_tr_s, c_tr_s, y_tr_s, M_small, k=5, lam=lam)
    cv_means.append(scores.mean())
    cv_stds.append(scores.std())

cv_means = np.array(cv_means)
cv_stds = np.array(cv_stds)

# Plot CV performance vs hyperparameter
plt.figure()
plt.errorbar(lambda_grid, cv_means, yerr=cv_stds, fmt='o-', capsize=5, markersize=8)
plt.xscale('log')
plt.xlabel(r'Regularization strength $\lambda$')
```

```
plt.ylabel('Cross-Validation AUC')
plt.title('Hyperparameter Tuning')
plt.grid(True, alpha=0.3)
plt.show()

best_lam_idx = np.argmax(cv_means)
print(f"Best  : {lambda_grid[best_lam_idx]:.3f} (CV AUC: {cv_means[best_lam_idx]:.4f} ± {cv_stds[best_lam_idx]:.4f})")
```

The error bars show the standard deviation across folds, indicating uncertainty in the CV estimate for each hyperparameter. Select the hyperparameter with highest mean CV performance.

2.7.3 Beyond AUC: Multiple Evaluation Metrics

AUC measures ranking quality but does not capture all aspects of model performance. Additional metrics provide complementary insights:

Log-Likelihood: Measures the probability the model assigns to observed outcomes. Higher is better.

$$\text{LL} = \sum_{(j,j') \in \mathcal{D}_{\text{test}}} \left[Y_{jj'} \log \sigma(V_j - V_{j'}) + (1 - Y_{jj'}) \log(1 - \sigma(V_j - V_{j'})) \right] \tag{2.18}$$

Calibration Error: Measures whether predicted probabilities match empirical frequencies. Bin predictions into intervals (e.g., [0.0, 0.1), [0.1, 0.2), …) and compare average predicted probability to observed frequency in each bin.

```
#| autorun: true
def compute_log_likelihood(V, r_idx, c_idx, y_obs):
    """Compute log-likelihood on test data."""
    s = V[r_idx] - V[c_idx]
    p = 1.0 / (1.0 + np.exp(-s))
    # Numerical stability: use log(p) and log(1-p) carefully
    ll = (y_obs * np.log(p + 1e-12) + (1 - y_obs) * np.log(1 - p + 1e-12)).sum()
    return ll

def compute_calibration_error(V, r_idx, c_idx, y_obs, n_bins=10):
    """Compute calibration error: difference between predicted prob and observed freq."""
    s = V[r_idx] - V[c_idx]
    p_pred = 1.0 / (1.0 + np.exp(-s))

    # Bin predictions
    bins = np.linspace(0, 1, n_bins + 1)
    cal_error = 0.0
```

```
    for i in range(n_bins):
        in_bin = (p_pred >= bins[i]) & (p_pred < bins[i+1])
        if in_bin.sum() > 0:
            avg_pred = p_pred[in_bin].mean()
            avg_obs = y_obs[in_bin].mean()
            cal_error += np.abs(avg_pred - avg_obs) * in_bin.sum()
    cal_error /= len(y_obs)
    return cal_error

# Compare multiple metrics for different regularization strengths
lambdas_compare = [0.0, 0.1, 1.0]
metrics_results = []

for lam in lambdas_compare:
    V_fit = fit_regularized_bt(r_tr_s, c_tr_s, y_tr_s, M_small, lam=lam)

    # Compute metrics on test data
    s_te = V_fit[r_te_s] - V_fit[c_te_s]
    auc_te = auc_from_scores(s_te, y_te_s)
    ll_te = compute_log_likelihood(V_fit, r_te_s, c_te_s, y_te_s)
    cal_err = compute_calibration_error(V_fit, r_te_s, c_te_s, y_te_s)

    metrics_results.append({
        'lambda': lam,
        'AUC': auc_te,
        'Log-Likelihood': ll_te,
        'Calibration Error': cal_err
    })

# Display results
print("\nMulti-Metric Comparison:")
print(f"{' ':<8} {'AUC':<8} {'Log-Lik':<12} {'Cal Error':<12}")
print("-" * 40)
for res in metrics_results:
    print(f"{res['lambda']:<8.2f} {res['AUC']:<8.4f} {res['Log-Likelihood']:<12.2f}
    ↪ {res['Calibration Error']:<12.4f}")
```

Different metrics may favor different models. AUC focuses on ranking, log-likelihood on probability estimates, and calibration on frequency matching. Use multiple metrics to gain a comprehensive view of model quality.

! Key Takeaway: Cross-Validation Best Practices

- **Use CV for hyperparameter tuning**, not test set evaluation (avoid overfitting to the test set)
- **Report mean ± std** across folds to quantify uncertainty
- **Stratified splitting**: For imbalanced data, ensure each fold has representative class proportions
- **Temporal splitting**: For sequential data (e.g., chess games over time), use time-based splits instead of random CV to avoid leaking future information into past predictions

– **Nested CV**: For unbiased performance estimates, use outer CV loop for evaluation and inner CV loop for hyperparameter selection

2.8 Optimization Methods

Gradient descent is the foundation for learning Bradley-Terry parameters, but modern optimization methods can accelerate convergence and improve final performance. We compare three widely-used optimizers and discuss when each is appropriate.

2.8.1 Beyond Vanilla Gradient Descent

Standard gradient descent updates parameters with a fixed learning rate: $V \leftarrow V + \eta \nabla \mathcal{L}(V)$. This has two limitations:

1. **Fixed step size**: Large η causes instability; small η slows convergence
2. **No momentum**: Each step ignores the optimization history

Modern optimizers address these issues through adaptive learning rates and momentum.

2.8.2 Adam Optimizer

Adam (Adaptive Moment Estimation) maintains running averages of both gradients and squared gradients, adapting the learning rate per parameter. The update rule is:

$$\begin{aligned} m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t && \text{(momentum)} \\ v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 && \text{(variance)} \\ \hat{m}_t &\leftarrow m_t/(1-\beta_1^t), \quad \hat{v}_t \leftarrow v_t/(1-\beta_2^t) && \text{(bias correction)} \\ V &\leftarrow V + \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} && \text{(parameter update)} \end{aligned} \tag{2.19}$$

where g_t is the gradient at step t, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are typical, and $\epsilon = 10^{-8}$ prevents division by zero.

```
#| autorun: true
def fit_bt_adam(r_idx, c_idx, y_obs, M, lr=0.1, epochs=200, beta1=0.9, beta2=0.999,
    eps=1e-8):
    """Fit Bradley-Terry using Adam optimizer."""
    V = np.zeros(M)
    m = np.zeros(M)  # first moment
    v = np.zeros(M)  # second moment

    loss_history = []
```

```
    for t in range(1, epochs + 1):
        # Compute gradient
        H = V[r_idx] - V[c_idx]
        p = 1.0 / (1.0 + np.exp(-H))
        err = y_obs - p
        grad = np.zeros(M)
        np.add.at(grad, r_idx, err)
        np.add.at(grad, c_idx, -err)

        # Adam update
        m = beta1 * m + (1 - beta1) * grad
        v = beta2 * v + (1 - beta2) * (grad ** 2)

        # Bias correction
        m_hat = m / (1 - beta1 ** t)
        v_hat = v / (1 - beta2 ** t)

        # Parameter update
        V += lr * m_hat / (np.sqrt(v_hat) + eps)

        # Log-likelihood for monitoring
        ll = (y_obs * np.log(p + 1e-12) + (1 - y_obs) * np.log(1 - p + 1e-12)).sum()
        loss_history.append(-ll)  # negative log-likelihood (loss)

    return V, np.array(loss_history)

# Compare GD vs Adam on the same problem
V_gd, loss_gd = [], []
V_adam, loss_adam = fit_bt_adam(r_tr_s, c_tr_s, y_tr_s, M_small, lr=0.1, epochs=150)

# Also fit with vanilla GD
V = np.zeros(M_small)
for epoch in range(150):
    H = V[r_tr_s] - V[c_tr_s]
    p = 1.0 / (1.0 + np.exp(-H))
    err = y_tr_s - p
    grad = np.zeros(M_small)
    np.add.at(grad, r_tr_s, err)
    np.add.at(grad, c_tr_s, -err)
    V += 0.05 * grad  # smaller LR for GD stability
    ll = (y_tr_s * np.log(p + 1e-12) + (1 - y_tr_s) * np.log(1 - p + 1e-12)).sum()
    loss_gd.append(-ll)

loss_gd = np.array(loss_gd)

# Plot convergence comparison
plt.figure()
plt.plot(loss_gd, label='Gradient Descent (lr=0.05)', alpha=0.8)
```

```
plt.plot(loss_adam, label='Adam (lr=0.1)', alpha=0.8)
plt.xlabel('Epoch')
plt.ylabel('Negative Log-Likelihood (Loss)')
plt.title('Optimization Comparison: GD vs Adam')
plt.legend()
plt.grid(True, alpha=0.3)
plt.yscale('log')
plt.show()

print(f"Final loss - GD: {loss_gd[-1]:.4f}, Adam: {loss_adam[-1]:.4f}")
```

Adam typically converges faster than vanilla gradient descent and is less sensitive to learning rate tuning. The adaptive per-parameter learning rates help in problems with varying gradient magnitudes across parameters.

2.8.3 Learning Rate Schedules

Instead of a fixed learning rate, **schedules** decay η over time to enable large initial steps (fast progress) followed by small refinements (precision):

- **Step decay**: $\eta_t = \eta_0 \cdot \gamma^{\lfloor t/k \rfloor}$ (reduce by factor γ every k epochs)
- **Exponential decay**: $\eta_t = \eta_0 e^{-\lambda t}$
- **Cosine annealing**: $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min})(1 + \cos(\pi t/T))$

Learning rate schedules are especially useful for training to convergence without extensive hyperparameter tuning.

> **Key Takeaway: Choosing an Optimizer**
>
> **When to use each optimizer**:
>
> - **Vanilla GD**: Simple problems, well-tuned learning rate available, or for pedagogical clarity
> - **Adam**: Default choice for most problems; robust to learning rate, fast convergence, handles sparse gradients well
> - **SGD with momentum**: When you need the theoretical guarantees of SGD (e.g., convergence proofs) but want faster practical convergence
>
> **Hyperparameter guidelines**: - Adam: $\alpha = 0.001$ to 0.1, default $\beta_1 = 0.9$, $\beta_2 = 0.999$ - GD: $\eta = 0.01$ to 0.1, may need careful tuning - Learning rate schedules: Helpful when training for many epochs

2.9 Real-World Application: LLM Preference Learning

We now apply all three parameter learning methods to a realistic preference learning scenario inspired by language model alignment. While production LLM alignment uses datasets like Stanford Human Preferences (SHP) or Anthropic's HH-RLHF, we construct a synthetic

dataset that captures key properties of real preference data: noisy labels, varying item quality, and response embeddings.

2.9.1 Dataset: Simulated LLM Response Preferences

We simulate a setting where a language model generates multiple responses to prompts, and human annotators provide pairwise preferences. Each response is represented by a learned embedding (e.g., from a pretrained model), and the true utility is a linear function of this embedding plus noise.

```
#| autorun: true
# Simulate LLM response preference data
rng_llm = np.random.default_rng(2024)

# Setting: 50 LLM responses across different prompts
n_responses = 50
embedding_dim = 8  # low-dimensional for interpretability

# Generate response embeddings (simulate learned representations)
# In reality, these would be from a language model's hidden states
response_embeddings = rng_llm.normal(0, 1, size=(n_responses, embedding_dim))
# Normalize embeddings
response_embeddings /= np.linalg.norm(response_embeddings, axis=1, keepdims=True)

# True utility weights (what humans actually care about)
# Positive weights favor certain embedding dimensions
true_weights = rng_llm.normal(0, 1, size=embedding_dim)
true_weights[0] = 2.0  # Strong preference for first dimension (e.g., helpfulness)
true_weights[1] = -1.5  # Dispreference for second dimension (e.g., verbosity)

# Compute true utilities
true_utilities = response_embeddings @ true_weights

# Generate pairwise comparisons with label noise (realistic!)
n_comparisons = 200
comparison_pairs = []
comparison_labels = []

for _ in range(n_comparisons):
    # Sample a pair of responses
    i, j = rng_llm.choice(n_responses, size=2, replace=False)
    comparison_pairs.append((i, j))

    # True win probability with Bradley-Terry
    p_i_wins = 1.0 / (1.0 + np.exp(-(true_utilities[i] - true_utilities[j])))

    # Add label noise: 10% of annotations are random
    if rng_llm.random() < 0.1:
```

```
        label = rng_llm.choice([0, 1])  # noisy label
    else:
        label = 1 if rng_llm.random() < p_i_wins else 0

    comparison_labels.append(label)

comparison_pairs = np.array(comparison_pairs)
comparison_labels = np.array(comparison_labels, dtype=float)

print(f"Generated {n_comparisons} comparisons over {n_responses} LLM responses")
print(f"True utility range: [{true_utilities.min():.2f}, {true_utilities.max():.2f}]")
print(f"Label noise: ~10% of comparisons are random")
```

This synthetic dataset mimics real LLM preference data: responses have embedding representations, utilities are learned functions of embeddings, and annotations contain noise.

> **! Label Noise: Random vs. Systematic**
>
> The 10% uniform label noise above is a simplification. In real annotation pipelines, noise is often **systematic** rather than random: annotators may have consistent biases (preferring verbose responses), may be influenced by presentation order, or may satisfice rather than carefully evaluate. Uniform noise attenuates the learned utilities toward zero (biasing toward equal preference) but preserves their *ranking*. Systematic noise, by contrast, can **reverse rankings**—if annotators consistently prefer item j for reasons unrelated to the true objective (e.g., surface fluency over factual accuracy), the MLE will learn the annotators' biases as if they were genuine preferences. This connects to the **inversion problem** discussed in Chapter 6: observed choices may not reflect underlying preferences. Robust estimation methods that model annotator-specific biases or down-weight unreliable annotations can mitigate this issue.

2.9.2 Applying All Three Learning Methods

We apply MLE, Bayesian inference, and online learning (Elo) to the same data and compare results.

```
#| autorun: true
# Split into train/test (80/20)
n_train = int(0.8 * n_comparisons)
train_idx = rng_llm.choice(n_comparisons, n_train, replace=False)
test_idx = np.array([i for i in range(n_comparisons) if i not in train_idx])

r_train = comparison_pairs[train_idx, 0]
c_train = comparison_pairs[train_idx, 1]
y_train = comparison_labels[train_idx]

r_test = comparison_pairs[test_idx, 0]
c_test = comparison_pairs[test_idx, 1]
y_test = comparison_labels[test_idx]
```

```

# Method 1: Maximum Likelihood with L2 Regularization
V_mle = fit_regularized_bt(r_train, c_train, y_train, n_responses, lam=0.05, lr=0.1,
 ↪  epochs=300)

# Method 2: Bayesian Inference via MCMC (Metropolis-Hastings)
def mcmc_bradley_terry(r, c, y, n_items, prior_std=1.0, n_samples=2000, burnin=500,
 ↪  step_size=0.15):
    """Sample from posterior of Bradley-Terry utilities using Metropolis-Hastings."""
    def log_posterior(V):
        diff = V[r] - V[c]
        ll = np.sum(y * diff - np.log(1 + np.exp(diff)))  # log-likelihood
        lp = -0.5 * np.sum(V**2) / prior_std**2  # log-prior: N(0, prior_std^2)
        return ll + lp

    # Initialize at MAP estimate for faster convergence
    V_current = fit_regularized_bt(r, c, y, n_items, lam=1.0/prior_std**2, lr=0.1,
 ↪  epochs=200)
    V_current = V_current - V_current.mean()
    log_p_current = log_posterior(V_current)
    samples = []

    for t in range(n_samples + burnin):
        # Propose: random walk on one randomly chosen utility
        V_proposed = V_current.copy()
        idx = np.random.randint(n_items)
        V_proposed[idx] += np.random.randn() * step_size
        V_proposed = V_proposed - V_proposed.mean()  # maintain centering

        # Accept/reject
        log_p_proposed = log_posterior(V_proposed)
        if np.log(np.random.rand()) < log_p_proposed - log_p_current:
            V_current, log_p_current = V_proposed, log_p_proposed

        if t >= burnin:
            samples.append(V_current.copy())

    return np.array(samples)

np.random.seed(42)
V_samples = mcmc_bradley_terry(r_train, c_train, y_train, n_responses,
                               prior_std=1.0, n_samples=1500, burnin=500)
V_bayes = V_samples.mean(axis=0)  # Posterior mean
V_bayes_std = V_samples.std(axis=0)  # Posterior uncertainty

# Method 3: Online Learning (Elo) - process comparisons sequentially
V_elo = np.zeros(n_responses)
K_factor = 0.1
for i, j, y in zip(r_train, c_train, y_train):
```

```
    p = 1.0 / (1.0 + np.exp(-(V_elo[i] - V_elo[j])))
    if y == 1.0:
        V_elo[i] += K_factor * (1 - p)
        V_elo[j] -= K_factor * (1 - p)
    else:
        V_elo[j] += K_factor * p
        V_elo[i] -= K_factor * p
    V_elo -= V_elo.mean()  # center

# Evaluate all three methods on test data
s_mle = V_mle[r_test] - V_mle[c_test]
s_bayes = V_bayes[r_test] - V_bayes[c_test]
s_elo = V_elo[r_test] - V_elo[c_test]

auc_mle = auc_from_scores(s_mle, y_test)
auc_bayes = auc_from_scores(s_bayes, y_test)
auc_elo = auc_from_scores(s_elo, y_test)

ll_mle = compute_log_likelihood(V_mle, r_test, c_test, y_test)
ll_bayes = compute_log_likelihood(V_bayes, r_test, c_test, y_test)
ll_elo = compute_log_likelihood(V_elo, r_test, c_test, y_test)

# Compare learned utilities to ground truth
# Align learned utilities to true utilities (up to affine transformation)
def align_utilities(V_learned, V_true):
    """Align learned to true utilities via least squares."""
    V_norm = (V_learned - V_learned.mean()) / (V_learned.std() + 1e-8)
    A = np.vstack([V_norm, np.ones_like(V_norm)]).T
    a, b = np.linalg.lstsq(A, V_true, rcond=None)[0]
    return a * V_norm + b

V_mle_aligned = align_utilities(V_mle, true_utilities)
V_bayes_aligned = align_utilities(V_bayes, true_utilities)
V_elo_aligned = align_utilities(V_elo, true_utilities)

corr_mle = np.corrcoef(true_utilities, V_mle_aligned)[0, 1]
corr_bayes = np.corrcoef(true_utilities, V_bayes_aligned)[0, 1]
corr_elo = np.corrcoef(true_utilities, V_elo_aligned)[0, 1]

# Display results
print("\n=== Method Comparison ===")
print(f"{'Method':<20} {'Test AUC':<12} {'Test LL':<12} {'Corr w/ True':<15}")
print("-" * 60)
print(f"{'MLE ( =0.05)':<20} {auc_mle:<12.4f} {ll_mle:<12.2f} {corr_mle:<15.4f}")
print(f"{'Bayesian (MCMC)':<20} {auc_bayes:<12.4f} {ll_bayes:<12.2f}
 ↪  {corr_bayes:<15.4f}")
print(f"{'Online (Elo K=0.1)':<20} {auc_elo:<12.4f} {ll_elo:<12.2f}
 ↪  {corr_elo:<15.4f}")

```

```
# Visualize learned vs true utilities (MLE and Elo)
plt.figure()
fig, axes = plt.subplots(1, 2)

for ax, V_learned, method, corr in zip(axes,
                                       [V_mle_aligned, V_elo_aligned],
                                       ['MLE', 'Elo'],
                                       [corr_mle, corr_elo]):
    ax.scatter(true_utilities, V_learned, alpha=0.6, s=30)
    ax.plot([true_utilities.min(), true_utilities.max()],
            [true_utilities.min(), true_utilities.max()],
            'r--', alpha=0.5)
    ax.set_xlabel('True Utility')
    ax.set_ylabel('Learned Utility')
    ax.set_title(f'{method} (r={corr:.3f})')
    ax.grid(True, alpha=0.3)

plt.show()

# Bayesian visualization with posterior uncertainty and marginal distributions
fig = plt.figure(layout='constrained')
gs = fig.add_gridspec(2, 2, width_ratios=[4, 1], height_ratios=[1, 4],
                      hspace=0.05, wspace=0.05)

ax_main = fig.add_subplot(gs[1, 0])
ax_top = fig.add_subplot(gs[0, 0], sharex=ax_main)
ax_right = fig.add_subplot(gs[1, 1], sharey=ax_main)

# Main scatter with error bars showing posterior std
ax_main.errorbar(true_utilities, V_bayes_aligned, yerr=V_bayes_std * 1.96,
                 fmt='o', alpha=0.5, markersize=4, capsize=2, elinewidth=0.5,
                 label='Posterior mean ± 95% CI')
ax_main.plot([true_utilities.min(), true_utilities.max()],
             [true_utilities.min(), true_utilities.max()], 'r--', alpha=0.5)
ax_main.set_xlabel('True Utility')
ax_main.set_ylabel('Learned Utility (Posterior Mean)')
ax_main.grid(True, alpha=0.3)
ax_main.legend(loc='upper left', fontsize=8)

# Marginal histogram/KDE for true utilities (top)
ax_top.hist(true_utilities, bins=20, density=True, alpha=0.7, color='steelblue')
ax_top.set_ylabel('Density')
ax_top.tick_params(labelbottom=False)

# Marginal showing posterior samples distribution (right)
# Sample a few posterior draws to show uncertainty
for sample in V_samples[::100]:  # every 100th sample
    sample_aligned = align_utilities(sample, true_utilities)
    ax_right.hist(sample_aligned, bins=20, density=True, alpha=0.1,
```

```
                   color='orange', orientation='horizontal')
ax_right.hist(V_bayes_aligned, bins=20, density=True, alpha=0.7,
              color='darkorange', orientation='horizontal', label='Posterior mean')
ax_right.set_xlabel('Density')
ax_right.tick_params(labelleft=False)

fig.suptitle(f'Bayesian Inference (r={corr_bayes:.3f})', y=0.98)
plt.show()
```

2.9.3 *Key Observations*

The three methods yield similar performance on this synthetic LLM preference task:

1. **MLE** is fastest and most scalable, suitable for large-scale applications
2. **Bayesian** inference via MCMC provides posterior distributions over utilities, enabling uncertainty quantification—the marginal plot shows how posterior samples vary, and error bars show 95% credible intervals
3. **Online (Elo)** learns incrementally, ideal for streaming data but may be less data-efficient than batch methods

All methods successfully recover utilities correlated with the ground truth despite 10% label noise, demonstrating robustness. In production LLM alignment:

- **Data scale**: Real datasets have 10K-100K+ comparisons
- **Cold start**: New responses have no comparison history; requires initialization strategies
- **Computational cost**: MCMC becomes expensive; MLE or online methods preferred
- **Temporal drift**: User preferences may evolve; online methods naturally adapt

CONNECTING TO DPO

Recall from that Direct Preference Optimization (DPO) for language models is equivalent to Bradley-Terry MLE where the "utility" is $\beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$. The MLE techniques developed in this chapter directly apply to DPO training: regularization prevents overfitting to preference data, and optimization methods (Adam, learning rate schedules) accelerate convergence.

COMMON PITFALL: DPO'S HIDDEN ASSUMPTIONS

DPO is often described as "reward-model-free," but it is not assumption-free. DPO's implicit reward is $r^*(y \mid x) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$, which means: (1) the recovered reward **depends on the reference policy** π_{ref} — different reference models yield different implicit rewards; (2) preferences are assumed to follow **Bradley-Terry** (i.i.d. Gumbel noise), so IIA violations in human judgments propagate silently; (3) the hyperparameter β controls the tradeoff between staying close to π_{ref} and optimizing for preferences — too small and the model barely moves; too large and it overfits to noisy labels. Always validate DPO-trained models against held-out preference data.

2.10 Summary

- This chapter develops three complementary approaches to estimating preference model parameters from observed comparison data, each with distinct tradeoffs in scalability, uncertainty quantification, and data requirements.
- **Maximum Likelihood Estimation (MLE)** finds point estimates by maximizing $\ell(V) = \sum \log \sigma(V_j - V_k)$ via gradient ascent. It is fast, scalable, and the foundation for DPO training in language model alignment.
- **Bayesian inference** via Markov Chain Monte Carlo (MCMC) provides full posterior distributions over parameters, enabling uncertainty quantification—but at substantially higher computational cost. The **Laplace approximation** offers a cheaper Gaussian approximation to the posterior, essential for GP-based preference models where the non-Gaussian likelihood makes exact inference intractable.
- **Online learning (Elo ratings)** processes comparisons one at a time as streaming data. Elo is precisely **stochastic gradient ascent** on the Bradley-Terry log-likelihood with step size K, bridging a 70-year-old rating system to modern optimization.
- **Regularization** (L2 penalty, early stopping) prevents overfitting, especially when comparisons are sparse. L2 regularization is equivalent to MAP estimation with a Gaussian prior, connecting frequentist and Bayesian perspectives.
- **Cross-validation** provides principled model selection—choosing regularization strength, model dimensionality, or kernel hyperparameters—by estimating out-of-sample performance without a separate test set.
- **Evaluation** requires multiple metrics: AUC measures ranking accuracy, log-likelihood measures calibration quality, and calibration error directly assesses whether predicted probabilities match observed frequencies.
- In practice, all three methods often recover similar utility estimates on well-specified problems; the choice depends on whether you need uncertainty (Bayesian), scalability (MLE), or streaming updates (online).

2.11 Quick Check

Test your understanding before proceeding to the exercises.

1. Why does the Bradley-Terry log-likelihood have a unique maximum when the comparison graph is strongly connected, but may not otherwise?

2. An Elo system uses $K = 32$. After a game where the favorite (predicted 80% win probability) loses, how many rating points does each player gain/lose? What if $K = 16$?

3. You fit a Bradley-Terry model with no regularization on a sparse dataset (50 items, 100 comparisons). Some items have never been compared. What happens to the MLE for those items? How does L2 regularization help?

4. A colleague says "our model has 95% AUC on the test set, so it's excellent." What additional evaluation would you recommend, and why might high AUC still mask poor performance?

2.12 Discussion Questions

- In the synthetic experiments, why is the Bayes optimal AUC less than 1.0 even though we know the true parameters? What does this reveal about the fundamental limits of prediction from noisy preference data?
- How does L2 regularization affect the bias-variance tradeoff in Bradley-Terry estimation? Under what data conditions (sample size, noise level, number of items) would you expect regularization to have the largest impact on test performance?
- When would you prefer online learning (Elo) over batch learning (MLE) for preference model estimation? Consider computational cost, data arrival patterns, and statistical efficiency. Can you design a hybrid approach that combines benefits of both?
- The Bayesian approach via MCMC provides posterior distributions over parameters, while MLE gives point estimates. Beyond uncertainty quantification, what practical advantages might the full posterior provide? How would you use posterior samples to make decisions (e.g., which items to present to users)?
- Cross-validation assumes that folds are exchangeable—that the data distribution is the same across all folds. For sequential preference data (e.g., chess games ordered in time, or LLM preferences from a changing user population), this assumption may be violated. How should cross-validation be adapted for temporal data? What are the risks of using standard k-fold CV on sequential data?
- In the LLM preference learning example, all three methods (MLE, Bayesian, Elo) achieved similar test AUC despite different training procedures. Under what circumstances would you expect the methods to diverge in performance? Consider data scale, noise level, model mis-specification, and computational budget.

2.13 Bibliographic Notes

Maximum likelihood estimation for preference models dates back to Bradley and Terry (1952)'s original work on paired comparisons. The connection to modern machine learning was established through applications in ranking (Herbrich, Minka, and Graepel 2006) and recommender systems (Koren, Bell, and Volinsky 2009). Hunter (2004) provided efficient MM algorithms for Bradley-Terry MLE that scale to large problems.

Bayesian inference for preference data has a long history in psychometrics and experimental design. Davidson (1970) developed Bayesian approaches for paired comparisons. Modern MCMC methods for Bradley-Terry models are surveyed in Caron and Doucet (2012). The connection between L2 regularization and Gaussian priors (MAP estimation) is standard in Bayesian machine learning (Murphy 2012).

The Elo rating system was introduced by Elo (1978) for chess rankings and has since been applied broadly to competitive games (Glickman 1999 introduced the Glicko system with uncertainty estimates), online platforms (Herbrich, Minka, and Graepel 2006 for Xbox matchmaking), and information retrieval. The interpretation of Elo as stochastic gradient descent on the Bradley-Terry log-likelihood clarifies its connection to machine learning (Weng and Lin 2011).

Regularization in statistical learning has roots in ridge regression (Hoerl and Kennard 1970) and has become central to modern machine learning (Hastie, Tibshirani, and Friedman 2009). Early stopping as implicit regularization was analyzed by Yao, Rosasco, and Caponnetto (2007). The bias-variance tradeoff provides the theoretical foundation for why regularization improves generalization (Geman, Bienenstock, and Doursat 1992).

Cross-validation was formalized by Stone (1974) and Geisser (1975) for model selection. Kohavi (1995) provided practical guidance for k-fold CV. Nested cross-validation to avoid selection bias was emphasized by Cawley and Talbot (2010). Temporal validation strategies for time-series data are discussed in Bergmeir, Hyndman, and Koo (2018).

Optimization methods for machine learning are surveyed in Ruder (2016). Adam was introduced by Kingma and Ba (2014) and has become the default optimizer for deep learning. Convergence analysis for gradient descent on convex problems (including logistic regression, which includes Bradley-Terry) is classical (Boyd and Vandenberghe 2004). Learning rate schedules and their impact on generalization are discussed in Loshchilov and Hutter (2016).

LLM alignment via preference learning gained prominence with Christiano et al. (2017)'s introduction of RLHF. The connection to Bradley-Terry and reward modeling was made explicit in Ouyang et al. (2022) (InstructGPT). Direct Preference Optimization (DPO) (Rafailov et al. 2023) showed that RLHF can be reformulated as Bradley-Terry MLE, eliminating the need for a separate reward model. Recent work explores calibration (Stiennon et al. 2020), robustness to noise (al. 2022), and scaling laws (Gao, Schulman, and Hilton 2022) for preference-based LLM training.

For further reading: Agresti (2002) provides comprehensive coverage of categorical data analysis including paired comparisons. Marden (1995) covers ranking models from a statistical perspective. Liu (2011) surveys learning-to-rank methods in information retrieval, many of which build on preference models.

Essential Reading

If you read only five papers from this chapter's references, make them these:

1. **Elo (1978)** — The Elo rating system: the most widely deployed preference learning algorithm in history
2. **Herbrich, Minka, and Graepel (2006)** — TrueSkill: Bayesian preference learning with uncertainty, the Elo successor
3. **Rafailov et al. (2023)** — DPO as Bradley-Terry MLE, unifying reward modeling and policy optimization
4. **Ouyang et al. (2022)** — InstructGPT: scaling RLHF to large language models
5. **Hastie, Tibshirani, and Friedman (2009)** — The statistical learning textbook covering regularization, cross-

validation, and model selection foundations

2.14 Further Reading

For readers who want to go deeper into the topics introduced in this chapter, we recommend the following:

- **Hunter (2004)**, "MM Algorithms for Generalized Bradley-Terry Models" — Efficient algorithms for MLE that scale to large comparison datasets, including extensions to ties and home-field advantage.
- **Caron and Doucet (2012)**, "Efficient Bayesian Inference for Generalized Bradley-Terry Models" — Modern MCMC and variational methods for Bayesian preference learning, including techniques for handling large-scale data.
- **Glickman (1999)**, "Parameter Estimation in Large Dynamic Paired Comparison Experiments" — The Glicko rating system, which extends Elo with per-player uncertainty estimates that decay with inactivity, addressing several limitations of fixed K-factor Elo.
- **Rasmussen and Williams (2006)**, *Gaussian Processes for Machine Learning*, Ch. 3 — Comprehensive treatment of the Laplace approximation and expectation propagation for GP classification, the theoretical foundation for the GP inference methods used in this chapter.
- **Gneiting and Raftery (2007)**, "Strictly Proper Scoring Rules, Prediction, and Estimation" — The definitive treatment of proper scoring rules and calibration, explaining why cross-entropy (log loss) is the natural loss for preference learning and how to assess whether learned probabilities are well-calibrated.
- **Ouyang et al. (2022)**, "Training Language Models to Follow Instructions with Human Feedback" — The InstructGPT paper that demonstrated RLHF at scale, showing how the MLE, Bayesian, and online methods in this chapter are applied to real LLM alignment.
- **Hastie, Tibshirani, and Friedman (2009)**, *The Elements of Statistical Learning* — The standard reference for regularization theory, bias-variance tradeoff, and cross-validation, providing the statistical foundations for the model selection techniques discussed here.

2.15 Exercises

Exercises are marked with difficulty levels: *() for introductory, () for intermediate, and ()* for challenging.

2.15.1 Gradient Derivation for Plackett-Luce (*)

The Plackett-Luce model extends Bradley-Terry to full rankings. Given a ranking $(j_1 \succ j_2 \succ \cdots \succ j_M)$, the likelihood is:

$$p(\text{ranking} \mid V) = \prod_{k=1}^{M-1} \frac{\exp(V_{j_k})}{\sum_{\ell=k}^{M} \exp(V_{j_\ell})} \tag{2.20}$$

(a) Write the log-likelihood for a single ranking as a function of utilities V.

(b) Derive the gradient $\frac{\partial \log p}{\partial V_m}$ for an arbitrary item m. *Hint: Consider three cases: m appears at position k, m appears after position k, and m does not appear in the ranking.*

(c) Implement gradient ascent for Plackett-Luce on simulated ranking data. Compare convergence to Bradley-Terry MLE—which converges faster and why?

2.15.2 L2 Regularized Bradley-Terry (**)

(a) Implement Bradley-Terry MLE with L2 regularization for a range of λ values. Generate synthetic data with $M = 20$ items and vary the number of comparisons from 50 to 500. Plot test AUC vs λ for each dataset size.

(b) Explain why the optimal λ decreases as the number of comparisons increases. At what rate does it decay?

(c) Derive the Hessian matrix of the regularized log-likelihood. Under what conditions is it positive definite (ensuring a unique global maximum)?

2.15.3 K-Fold Cross-Validation Implementation (**)

(a) Implement 5-fold cross-validation for Bradley-Terry estimation from scratch (without using sklearn or similar libraries). Your implementation should:

- Randomly partition comparison data into 5 folds
- Train on 4 folds, validate on 1 fold
- Repeat for all 5 folds
- Return mean and standard deviation of validation AUC

(b) Compare your CV implementation to a single 80/20 train/test split over 20 random seeds. Which provides a more stable performance estimate?

(c) Extend your implementation to stratified CV: ensure each fold has approximately equal proportions of wins for each item. Why might stratification improve CV estimates for imbalanced data?

2.15.4 Learning Rate Sensitivity Analysis (*)

(a) For Bradley-Terry MLE on synthetic data ($M = 15$ items, 100 comparisons), experiment with learning rates $\eta \in \{0.001, 0.01, 0.05, 0.1, 0.5, 1.0\}$. Plot the training loss curve for each learning rate.

(b) Identify the learning rate that achieves the lowest final loss. What happens with learning rates that are too large? Too small?

(c) Implement a learning rate schedule (e.g., exponential decay $\eta_t = \eta_0 \cdot 0.95^t$) and compare to fixed learning rate. Does the schedule improve final performance?

2.15.5 MCMC Diagnostics (**)

The Metropolis-Hastings implementation in produces a chain of samples. Assess convergence quality:

(a) Implement the **effective sample size** (ESS) diagnostic: ESS estimates how many independent samples the chain is equivalent to, accounting for autocorrelation. For a chain $\{V^{(t)}\}_{t=1}^{T}$, the ESS for parameter V_j is approximately:

$$\text{ESS}_j = \frac{T}{1 + 2\sum_{k=1}^{K} \rho_k} \tag{2.21}$$

where ρ_k is the autocorrelation at lag k.

(b) Run the MH algorithm from with different proposal step sizes $\tau \in \{0.01, 0.05, 0.1, 0.5\}$. Plot ESS vs τ. What happens when τ is too small? Too large?

(c) Implement trace plots for multiple chains (run MH 3 times with different initializations). Do all chains converge to the same stationary distribution?

2.15.6 Optimization Method Comparison (**)

(a) Implement gradient descent with momentum: $m_t \leftarrow \beta m_{t-1} + \nabla \mathcal{L}(V_t)$, $V_{t+1} \leftarrow V_t + \eta m_t$ where $\beta \in [0, 1)$ controls momentum.

(b) On the same synthetic Bradley-Terry problem, compare four optimizers: vanilla GD, GD with momentum ($\beta = 0.9$), Adam, and RMSprop. Plot convergence curves (loss vs iteration).

(c) For each optimizer, find the best learning rate via grid search. Which optimizer is most sensitive to learning rate tuning?

2.15.7 Convergence Proof (***)

Prove that gradient ascent on the Bradley-Terry log-likelihood converges to the global maximum (assuming the maximum exists).

(a) Show that the Bradley-Terry log-likelihood is strictly concave when the comparison graph is strongly connected (every pair of items is connected by a directed path of comparisons).

(b) Prove that gradient ascent with sufficiently small step size η converges to the unique global maximum. *Hint: Use the fact that the gradient vanishes only at the maximum for strictly concave functions.*

(c) What happens when the comparison graph is not strongly connected? Construct an example with multiple disconnected components and characterize the set of maximum likelihood estimators.

2.15.8 Calibration Metrics (*)

(a) Implement a calibration plot: bin predicted probabilities into 10 intervals $[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$, and for each bin, compute the average predicted probability and the empirical frequency of wins.

(b) Generate synthetic Bradley-Terry data and fit two models: one correctly specified (Bradley-Terry) and one mis-specified (assume all items have equal utility). Plot calibration curves for both. Which is better calibrated?

(c) Define the Expected Calibration Error (ECE): $\text{ECE} = \sum_{b=1}^{B} \frac{|B_b|}{N} |\bar{p}_b - \bar{y}_b|$ where $\bar{p}_b$ is the average predicted probability in bin b, $\bar{y}_b$ is the empirical frequency, and $|B_b|$ is the number of predictions in bin b. Compute ECE for your models.

2.15.9 Online to Batch Convergence (*)

(a) Implement Elo with a decreasing K-factor: $K_t = K_0/\sqrt{t}$ where t is the iteration number. This is known as the Robbins-Monro condition for stochastic approximation.

(b) On synthetic Bradley-Terry data, run Elo with decreasing K_t and compare the final learned parameters to batch MLE. How close do they converge?

(c) Prove (or argue informally) that Elo with $K_t = \eta/\sqrt{t}$ converges to the MLE in the limit as $t \to \infty$. Under what conditions does this hold?

2.15.10 Hyperparameter Tuning with Nested Cross-Validation (***)

To obtain an unbiased estimate of generalization performance when hyperparameters are selected via CV, we use nested (double) CV:

- **Outer loop**: K-fold CV to estimate generalization
- **Inner loop**: For each outer fold, use K-fold CV on the training data to select hyperparameters

(a) Implement nested 5-fold CV for Bradley-Terry with L2 regularization. The inner loop tunes λ, the outer loop estimates test AUC.

(b) Compare the nested CV test AUC estimate to: (i) a single train/test split with CV on training data for λ selection, and (ii) CV test AUC when λ is selected using the outer CV test set (data leakage). Which is more optimistic? Pessimistic?

(c) Why is nested CV important? Give an example where selecting hyperparameters on the same data used for final evaluation leads to overly optimistic performance estimates.

2.15.11 Real-World Dataset Application (***)

Apply the methods from this chapter to a real preference dataset:

(a) Obtain a dataset such as the Jester jokes dataset (user ratings), a subset of MovieLens, or chess game outcomes from Lichess. Describe the data: number of items, number of comparisons, sparsity.

(b) Apply all three methods (MLE with regularization, Bayesian inference, online Elo) and compare their performance using multiple metrics (AUC, log-likelihood, calibration error). Use cross-validation to tune hyperparameters.

(c) Visualize the learned item parameters. Do they agree with intuition (e.g., highly-rated movies have high utility)? Identify the top-5 and bottom-5 items according to each method—do the methods agree?

(d) Discuss practical challenges encountered: computational cost, cold-start items, missing data, temporal effects. How would you address these in a production system?

2.15.12 Bias-Variance Tradeoff Demonstration (**)

(a) Generate synthetic Bradley-Terry data with $M = 10$ items and vary the training set size $n \in \{20, 50, 100, 200, 500\}$ comparisons. For each n and several regularization strengths λ, fit models and compute:

- **Bias**: Average difference between learned and true parameters
- **Variance**: Standard deviation of learned parameters across 50 random datasets

(b) Plot bias and variance vs λ for each training set size. Verify that as λ increases, bias increases and variance decreases.

(c) Plot test AUC vs λ and identify the optimal λ for each n. How does the optimal regularization strength change with dataset size?

References

Agresti, Alan. 2002. *Categorical Data Analysis*. 2nd ed. New York: John Wiley & Sons.

al., Yuntao Bai et. 2022. "Constitutional AI: Harmlessness from AI Feedback." https://arxiv.org/abs/2212.08073.

Bergmeir, Christoph, Rob J. Hyndman, and Bonsoo Koo. 2018. "A Note on the Validity of Cross-Validation for Evaluating Autoregressive Time Series Prediction." *Computational Statistics & Data Analysis* 120: 70–83.

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

Bradley, Ralph Allan, and Milton E Terry. 1952. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45.

Caron, François, and Arnaud Doucet. 2012. "Efficient Bayesian Inference for Generalized Bradley-Terry Models." *Journal of Computational and Graphical Statistics* 21 (1): 174–96.

Cawley, Gavin C., and Nicola L. C. Talbot. 2010. "On over-Fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation." *Journal of Machine Learning Research* 11: 2079–2107.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. "Deep Reinforcement Learning from Human Preferences." *Advances in Neural Information Processing Systems* 30.

Davidson, Roger R. 1970. "On Extending the Bradley-Terry Model to Accommodate Ties in Paired Comparison Experiments." *Journal of the American Statistical Association* 65 (329): 317–28.

Elo, Arpad E. 1978. *The Rating of Chessplayers, Past and Present*. New York: Arco Publishing.

Ford Jr, Lester R. 1957. "Solution of a Ranking Problem from Binary Comparisons." *The American Mathematical Monthly* 64 (8P2): 28–33.

Gao, Leo, John Schulman, and Jacob Hilton. 2022. "Scaling Laws for Reward Model Overoptimization." *arXiv Preprint arXiv:2210.10760*.

Geisser, Seymour. 1975. "The Predictive Sample Reuse Method with Applications." *Journal of the American Statistical Association* 70 (350): 320–28.

Geman, Stuart, Elie Bienenstock, and Rene Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4 (1): 1–58.

Glickman, Mark E. 1999. "Parameter Estimation in Large Dynamic Paired Comparison Experiments." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 48 (3): 377–94. https://doi.org/10.1111/1467-9876.00159.

Gneiting, Tilmann, and Adrian E Raftery. 2007. "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association* 102 (477): 359–78.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd ed. New York: Springer.

Herbrich, Ralf, Tom Minka, and Thore Graepel. 2006. "TrueSkill: A Bayesian Skill Rating System." In *Advances in Neural Information Processing Systems*, 19:569–76.

Hoerl, Arthur E., and Robert W. Kennard. 1970. "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics* 12 (1): 55–67.

Hunter, David R. 2004. "MM Algorithms for Generalized Bradley-Terry Models." *The Annals of Statistics* 32 (1): 384–406. https://doi.org/10.1214/aos/1079120141.

Kingma, Diederik P., and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization." *arXiv Preprint arXiv:1412.6980*.

Kohavi, Ron. 1995. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection" 14 (2): 1137–45.

Koren, Yehuda, Robert Bell, and Chris Volinsky. 2009. "Matrix Factorization Techniques for Recommender Systems." *Computer* 42 (8): 30–37.

Liu, Tie-Yan. 2011. *Learning to Rank for Information Retrieval*. Springer.

Loshchilov, Ilya, and Frank Hutter. 2016. "SGDR: Stochastic Gradient Descent with Warm Restarts." *arXiv Preprint arXiv:1608.03983*.

Marden, John I. 1995. *Analyzing and Modeling Rank Data*. Chapman & Hall.

Murphy, Kevin P. 2012. *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press.

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, et al. 2022. "Training Language Models to Follow Instructions with Human Feedback." *Advances in Neural Information Processing Systems* 35: 27730–44.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." https://arxiv.org/abs/2305.18290.

Rasmussen, Carl Edward, and Christopher K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press.

Ruder, Sebastian. 2016. "An Overview of Gradient Descent Optimization Algorithms." *arXiv Preprint arXiv:1609.04747*.

Stiennon, Nisan, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. "Learning to Summarize from Human Feedback." *Advances in Neural Information Processing Systems* 33.

Stone, M. 1974. "Cross-Validatory Choice and Assessment of Statistical Predictions." *Journal of the Royal Statistical Society: Series B (Methodological)* 36 (2): 111–33. https://doi.org/10.1111/J.2517-6161.1974.TB00994.X.

Weng, Ruby C., and Chih-Jen Lin. 2011. "A Bayesian Approximation Method for Online Ranking." *Journal of Machine Learning Research* 12: 267–300.

Yao, Yuan, Lorenzo Rosasco, and Andrea Caponnetto. 2007. "On Early Stopping in Gradient Descent Learning." *Constructive Approximation* 26 (2): 289–315.

3 Elicitation

Intended Learning Outcomes

By the end of this chapter you will be able to:

- **Define** the measurement problem for user preference elicitation and distinguish it from the learning problem in Chapter 3.
- **Derive** the Fisher information for Rasch/Bradley-Terry models and explain why items with $p \approx 0.5$ are most informative.
- **Compare** A-optimal, D-optimal, and E-optimal design criteria for selecting informative queries.
- **Apply** Laplace approximation to update posterior beliefs about user preferences in real-time.
- **Implement** active selection policies that outperform random sampling in sample efficiency.
- **Extend** Fisher information analysis to non-linear scoring functions via local linearization.
- **Apply** information-theoretic acquisition functions to actively select queries for Gaussian Process preference models, building on the GP foundations from Chapters 2-3.
- **Explain** how D-optimal design applies to Direct Preference Optimization (DPO) for LLM alignment and derive the ADPO algorithm.
- **Analyze** active preference learning in robotics, including trajectory comparison and reward function inference.

3.1 Chapter Overview

Chapter 1 developed models for how preferences arise from latent utilities. Chapter 2 showed how to estimate these utilities from observed comparisons. This chapter addresses the complementary question: **given that human feedback is expensive, how should we choose which comparisons to request?**

This is the *measurement problem* in preference elicitation. Unlike the learning problem (which assumes data is given), the measurement problem actively designs experiments to maximize information gained per query. Consider training an LLM via RLHF: collecting 100,000 human preference labels is expensive and time-consuming. If we could achieve the same alignment quality with 20,000 strategically selected comparisons, we would save 80% of annotation cost.

The central insight is that **not all comparisons are equally informative**. Asking whether humans prefer "Hello!" to "Hi there!" reveals little about model quality. Comparing two substantively different responses to a complex prompt provides more signal. The mathematical framework of *Fisher information* and *optimal experimental design* formalizes this intuition.

Chapter Structure. We build from simple to complex settings:

1. **Scalar preferences** (): The Rasch model and Fisher information
2. **Multi-dimensional preferences** (): Factor models and optimal design criteria (A/D/E-optimal)
3. **Preference functions** (): Linear and non-linear scoring functions
4. **Nonparametric models** (): GP-based active learning with information-theoretic acquisition
5. **LLM alignment** (): Active DPO for large language models
6. **Applications**: Metric elicitation () and robotics ()

Connections to Prior Chapters:

Concept	Chapter 1-2	This Chapter
Bradley-Terry model	Likelihood, MLE ()	Fisher information, query selection
Factor models $U^\top V$	Parameter estimation ()	D-optimal pair selection
GP preference models	Laplace inference ()	Information-gain acquisition
DPO loss	Objective function ()	ADPO active selection

Historical Notes. Active learning for preferences has roots in multiple fields:

- **Psychometrics (1960s)**: Computer-adaptive testing used Fisher information to select test items matching student ability
- **Optimal experimental design (1970s)**: Fedorov and Kiefer developed A/D/E-optimality for linear models
- **Bayesian experimental design (1990s)**: Chaloner & Verdinelli unified information-theoretic approaches
- **RLHF/Active DPO (2020s)**: Applying these ideas to LLM alignment at scale

> "The purpose of optimal design is to achieve the desired precision with minimum cost." — V.V. Fedorov (1972)

3.2 Measurement of User Preference Vector

When items' appeals V_j are known or pre-calibrated, we can learn a new user's appetite U quickly by choosing informative items online. Asking only easy or only hard items gives near-deterministic responses that teach little. The key tool for quantifying "informativeness" is Fisher information.

Definition: Fisher Information

Definition 3.1. For a parametric model $p(Y \mid \theta)$, the **Fisher Information** about parameter θ is:

$$\mathcal{I}(\theta) = \mathbb{E}\left[-\frac{\partial^2}{\partial\theta^2}\log p(Y \mid \theta)\right] = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log p(Y \mid \theta)\right)^2\right]$$

Fisher Information quantifies how much a single observation reveals about θ. Higher Fisher Information means more precise estimation from each observation. The Cramér-Rao bound states that the variance of any unbiased estimator is at least $\mathcal{I}(\theta)^{-1}$.

The Rasch model makes this precise. Recall from that for a candidate item j, the Bernoulli success probability is $p_j(U) = \sigma(U + V_j)$. The Fisher information about U from a single response is

$$\mathcal{I}_j(U) = \mathbb{E}\left[-\frac{\partial^2}{\partial U^2} \log p(Y \mid U, V_j)\right] = p_j(U)(1 - p_j(U)). \tag{3.1}$$

Proposition: Optimal Item Difficulty

Proposition 3.1. *For the Rasch model with $p_j(U) = \sigma(U + V_j)$, the Fisher Information $\mathcal{I}_j(U) = p_j(1 - p_j)$ is maximized when $p_j(U) = 0.5$, which occurs when item difficulty $-V_j$ equals user ability U.*

Proof

The function $f(p) = p(1 - p)$ is a downward parabola with roots at $p = 0$ and $p = 1$. Taking the derivative: $f'(p) = 1 - 2p = 0 \Rightarrow p = 0.5$. At this point, $f(0.5) = 0.25$, which is the maximum. Since $p_j(U) = \sigma(U + V_j) = 0.5$ when $U + V_j = 0$, the optimal item has $V_j = -U$.

Worked Example: Fisher Information for Item Selection

A user has estimated ability $\hat{U} = 1.0$. We have three candidate items with appeals $V_1 = -2.0$ (hard), $V_2 = -1.0$ (matched), and $V_3 = 1.0$ (easy). Which item should we ask next?

Step 1. Compute predicted success probabilities:

$$p_1 = \sigma(1.0 + (-2.0)) = \sigma(-1.0) \approx 0.269$$

$$p_2 = \sigma(1.0 + (-1.0)) = \sigma(0.0) = 0.500$$

$$p_3 = \sigma(1.0 + 1.0) = \sigma(2.0) \approx 0.881$$

Step 2. Compute Fisher information for each item:

$$\mathcal{I}_1 = p_1(1 - p_1) = 0.269 \times 0.731 \approx 0.197$$

$$\mathcal{I}_2 = p_2(1 - p_2) = 0.500 \times 0.500 = 0.250$$

$$\mathcal{I}_3 = p_3(1 - p_3) = 0.881 \times 0.119 \approx 0.105$$

Step 3. Select the most informative item: $j^* = \arg\max_j \mathcal{I}_j = 2$ (the matched item).

Interpretation: Item 2, whose difficulty $-V_2 = 1.0$ matches the user's ability $\hat{U} = 1.0$, is most informative because it produces responses closest to 50-50. The easy item (V_3) is least informative — the user almost certainly

succeeds, so the response tells us little about their ability. This is the principle behind *computerized adaptive testing*: the test "follows" the student, always selecting questions near their ability level.

This result has a natural interpretation: items that are too easy ($p \approx 1$) or too hard ($p \approx 0$) yield near-deterministic responses that teach little about the user. Items matched to the user's ability ($p \approx 0.5$) are maximally informative. So the best next item is one whose difficulty ($-V_j$) is close to the current estimate of U. Active elicitation policies therefore "hover" around the user's current location, rapidly shrinking uncertainty. Place a Gaussian prior $U \sim \mathcal{N}(0, \sigma_0^2)$. After observing independent items $\mathcal{J}$, a Laplace (or IRLS) update for U uses the score and observed information

$$S(U) = \sum_{j \in \mathcal{J}} (y_j - p_j(U)), \qquad \mathcal{I}(U) = \sum_{j \in \mathcal{J}} p_j(U)(1 - p_j(U)) + \tau_0, \quad \tau_0 = \sigma_0^{-2}. \tag{3.2}$$

A Newton step is $\hat{U} \leftarrow \hat{U} + S(\hat{U})\mathcal{I}(\hat{U})^{-1}$. The approximate posterior variance is $\widehat{\mathrm{Var}}(U) \approx \mathcal{I}(\hat{U})^{-1}$. An item-selection rule that maximizes expected **incremental** information picks $j_t = \arg\max_j \; p_j(\hat{U}_{t-1})(1 - p_j(\hat{U}_{t-1}))$. For evaluation, we can use reliability. With population variance $\mathrm{Var}(U^\star) = \sigma_U^2$, reliability is the fraction of total variance explained by the test rather than estimation noise:

$$\mathrm{Rel} \approx 1 - \frac{1}{N} \sum_{i=1}^{N} \frac{\widehat{\mathrm{Var}}(U_i)}{\sigma_U^2}. \tag{3.3}$$

Below, the code simulates many users, a bank of calibrated items, and compares an Fisher policy to a random policy. We track reliability after each query count t.

First, we set up the simulation with N users, M calibrated items, and define helper functions for Bayesian updating:

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| autorun: true
sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))
rng = np.random.default_rng(42)

# Simulation parameters
N = 200           # number of users
M = 200           # number of calibrated items
T = 40            # queries per user
sigma_U = 1.0     # population SD of user abilities
sigma0 = 1.0      # prior SD for each user

# Generate true user abilities and item difficulties
U_true = rng.normal(0, sigma_U, size=N)
V = np.sort(rng.normal(0, 1.0, size=M))

print(f"Simulating {N} users, {M} items, {T} queries each")
print(f"User abilities range: [{U_true.min():.2f}, {U_true.max():.2f}]")
print(f"Item difficulties range: [{V.min():.2f}, {V.max():.2f}]")
```

Next, we define the Bayesian update rule and the two item selection policies—Fisher-active (maximizing information) and random:

```
#| autorun: true
def new_user_state():
    """Initialize user state with prior."""
    return {"U_hat": 0.0, "tau": 1.0/(sigma0**2), "var": sigma0**2}

def update_user(state, vj, y):
    """Single Newton-IRLS update after observing response y to item j."""
    U, tau = state["U_hat"], state["tau"]
    p = sigmoid(U + vj)
    S, I = (y - p), p * (1 - p)  # score and Fisher info
    state["U_hat"] = U + S / (I + tau + 1e-12)
    state["tau"] = tau + I
    state["var"] = 1.0 / (state["tau"] + 1e-12)

def choose_item_fisher(state, V, asked_mask):
    """Select item maximizing Fisher information at current estimate."""
    cand = np.where(~asked_mask)[0]
    if cand.size == 0: return None
    vals = sigmoid(state["U_hat"] + V[cand])
    info = vals * (1.0 - vals)
    return cand[np.argmax(info + 1e-6 * rng.random(size=cand.size))]

def choose_item_random(state, V, asked_mask):
    """Select item uniformly at random."""
    cand = np.where(~asked_mask)[0]
    return rng.choice(cand) if cand.size > 0 else None
```

```

print("Update and selection functions defined.")
```

Now we run the simulation, comparing both policies across all users:

```
#| autorun: true
def run_policy(policy_fn, V, T):
    """Run policy for all users and track MSE/reliability curves."""
    states = [new_user_state() for _ in range(N)]
    asked = np.zeros((N, M), dtype=bool)
    mse_curve, rel_curve = [], []

    for t in range(1, T + 1):
        for i in range(N):
            j = policy_fn(states[i], V, asked[i])
            if j is None: continue
            y = 1 if rng.random() < sigmoid(U_true[i] + V[j]) else 0
            update_user(states[i], V[j], y)
            asked[i, j] = True

        U_hat = np.array([s["U_hat"] for s in states])
        Var_hat = np.array([s["var"] for s in states])
        mse_curve.append(np.mean((U_hat - U_true)**2))
        rel_curve.append(1.0 - np.mean(Var_hat) / sigma_U**2)

    return np.array(mse_curve), np.array(rel_curve)

# Run both policies
mse_act, rel_act = run_policy(choose_item_fisher, V, T)
mse_rnd, rel_rnd = run_policy(choose_item_random, V, T)
print("Simulation complete!")
```

Finally, we visualize the results comparing Fisher-active vs random item selection:

```
#| autorun: true
fig, axes = plt.subplots(1, 2)

axes[0].plot(np.arange(1, T+1), mse_rnd, 'o-', label="Random", alpha=0.7)
axes[0].plot(np.arange(1, T+1), mse_act, 's-', label="Fisher-active", alpha=0.7)
axes[0].set_xlabel("Queries per user"); axes[0].set_ylabel("MSE of U")
axes[0].set_title("Estimation Error vs Queries"); axes[0].legend(); axes[0].grid(True,
↪  alpha=0.3)

axes[1].plot(np.arange(1, T+1), rel_rnd, 'o-', label="Random", alpha=0.7)
axes[1].plot(np.arange(1, T+1), rel_act, 's-', label="Fisher-active", alpha=0.7)
axes[1].set_xlabel("Queries per user"); axes[1].set_ylabel("Reliability")
```

```
axes[1].set_title("Reliability vs Queries"); axes[1].legend(); axes[1].grid(True,
    alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Final MSE  | Random: {mse_rnd[-1]:.4f} | Fisher: {mse_act[-1]:.4f}")
print(f"Final Rel. | Random: {rel_rnd[-1]:.3f}  | Fisher: {rel_act[-1]:.3f}")
```

KEY TAKEAWAY: ACTIVE BEATS RANDOM

Fisher-active item selection achieves higher reliability with fewer queries than random selection. The gain comes from targeting items near $p = 0.5$, where each response is maximally informative. This principle—**measure where uncertainty is highest**—underlies all active learning methods in this chapter.

ASSUMPTION: KNOWN ITEM PARAMETERS

The Fisher information framework above assumes item parameters (V_j, Z_j) are **known or pre-calibrated** before elicitation begins. In practice, item parameters must often be estimated jointly with user preferences—this is the **cold-start problem** in recommender systems. When item parameters are uncertain, errors in item calibration propagate into suboptimal query selection: the system may ask about items it *thinks* are informative but are not. Joint estimation of user and item parameters requires different approaches, such as alternating optimization or fully Bayesian models that maintain uncertainty over both.

3.2.1 Factor Models: Multi-Dimensional Preferences

In practice, a single "ability" parameter may not capture the diversity of user preferences. Factor models generalize Rasch by giving each user a K-dimensional embedding $U_i \in \mathbb{R}^K$ and each item a K-dimensional embedding $V_j \in \mathbb{R}^K$, plus optional bias Z_j. The linear predictor is $H_{ij} = U_i^\top V_j + Z_j$ where $p(Y_{ij} = 1 \mid U_i, V_j, Z_j) = \sigma(H_{ij})$.

A concrete instance is **performance metric elicitation** (Hiranandani et al. 2019). In binary classification, practitioners often have implicit preferences over classifiers that don't match standard metrics like accuracy or F1—for instance, in medical diagnosis, the cost of a false negative (missed disease) may differ greatly from a false positive (unnecessary treatment). This is a $K = 2$ factor model where "items" are classifiers with known confusion matrices $V_\theta = (\text{TP}_\theta, \text{TN}_\theta)$, and the "user preference" is the unknown metric weights $\mathbf{m} = (m_{11}, m_{00})$. The utility $\mathbf{m}^\top V_\theta = m_{11} \cdot \text{TP}_\theta + m_{00} \cdot \text{TN}_\theta$ is exactly the factor model form with known item parameters.

For a given user U and an item j with known parameters, the Bernoulli log-likelihood is

$$\ell(U; y, j) = y \log \sigma(H) + (1 - y) \log(1 - \sigma(H)) \tag{3.4}$$

The gradient with respect to U is $\nabla_U \ell = (y - \sigma(H))V_j$. The expected Fisher Information per item is the negative expected Hessian:

$$\mathcal{I}_j(U) = \mathbb{E}[\nabla_U \ell \nabla_U \ell^\top] = \sigma(H)(1 - \sigma(H))V_j V_j^\top. \tag{3.5}$$

So each queried item contributes a rank-1 matrix in the direction of V_j, scaled by the variance term $\sigma(H)(1 - \sigma(H))$. For a sequence of items j_t, the cumulative information is

$$\mathcal{I}(U) = \sum_t \sigma(H_{j_t})(1 - \sigma(H_{j_t}))V_{j_t} V_{j_t}^\top. \tag{3.6}$$

Assumption: Correct Model Specification

Fisher information quantifies the informativeness of a query **under the assumed model**. If the model is misspecified—for example, if preferences are not logistic, or if the factor structure has the wrong dimensionality K—then queries optimized for the Fisher criterion may not be optimal for learning the true preference structure. Under misspecification, information-theoretic criteria such as mutual information or Bayesian approaches that integrate over model uncertainty can be more robust. In practice, it is worth checking sensitivity to the assumed model by comparing active learning performance across different model specifications.

In multiple dimensions, information is a matrix. Classical optimal design offers various criteria for selecting informative queries:

Definition: Optimal Design Criteria

Definition 3.2. Given a Fisher Information matrix $\mathcal{I}(U)$, the classical **optimal experimental design** criteria are:

- **A-Optimal (Average)**: Minimize $\text{tr}(\mathcal{I}^{-1})$—the average variance of parameter estimates
- **D-Optimal (Determinant)**: Maximize $\det(\mathcal{I})$—the volume shrinkage of the posterior ellipsoid
- **E-Optimal (Eigenvalue)**: Maximize $\lambda_{\min}(\mathcal{I})$—the precision in the worst-case direction

All three reduce uncertainty, but emphasize different aspects: A-optimal targets average precision, D-optimal targets overall information volume, and E-optimal targets the least-informed direction.

Common Pitfall: Confusing A-Optimal, D-Optimal, and E-Optimal

These three criteria **optimize different objectives** and can select different queries. A-optimal minimizes average variance (good when all parameters matter equally), D-optimal maximizes the determinant of the information matrix (shrinks the posterior ellipsoid volume — good for overall uncertainty), and E-optimal maximizes the minimum eigenvalue (guards against worst-case directions). A common mistake is to treat them as interchangeable. In practice: D-optimal naturally encourages **diversity** in selected items (redundant items give diminishing information), while E-optimal is **conservative** (focuses all effort on the least-informed direction). Choose the criterion based on your downstream task — if you need accurate *ranking* of all items, A-optimal is natural; if you need a single *best* decision, E-optimal may be preferable.

Adaptive measurement picks the next item j to maximize an acquisition function such as

$\det(\mathcal{I} + \mathcal{I}_j)$. We also use reliability as the evaluation metric here. If Σ_U is the population covariance and $\widehat{\Sigma}_{\text{err}}$ the average posterior covariance, then Reliability $= 1 - \text{tr}(\widehat{\Sigma}_{\text{err}})\text{tr}(\Sigma_U)^{-1}$. This generalizes the Rasch reliability to multi-dimensional latent traits. Below is a toy simulation with 2-dimensional users and items, comparing random item selection vs D-optimal Fisher active selection:

First, set up the factor model simulation with K-dimensional user and item embeddings:

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))
rng = np.random.default_rng(0)

N, M, K, T = 100, 300, 2, 30  # users, items, factors, queries
U_true = rng.normal(0, 1, size=(N, K))
V = rng.normal(0, 1, size=(M, K))
Z = rng.normal(0, 0.3, size=(M,))

def new_user_state():
    return {"U_hat": np.zeros(K), "Sigma": np.eye(K) * 2.0}

print(f"Factor model: {N} users, {M} items, K={K} dimensions")
```

Define the Bayesian update and D-optimal selection:

```
#| autorun: true
def update_user(state, vj, zj, y):
    """Update posterior with one item response via Laplace approximation."""
    U, S = state["U_hat"], state["Sigma"]
    p = sigmoid(U @ vj + zj)
    I_j = p * (1 - p) * np.outer(vj, vj)
    Sigma_new = np.linalg.inv(np.linalg.inv(S) + I_j)
    state["U_hat"] = U + Sigma_new @ ((y - p) * vj)
    state["Sigma"] = Sigma_new

def choose_item_Dopt(state, asked_mask, V, Z):
    """Select item maximizing D-optimal log-det gain."""
    Prec = np.linalg.inv(state["Sigma"])
    cand = np.where(~asked_mask)[0]
    scores = []
    for j in cand:
        W = sigmoid(state["U_hat"] @ V[j] + Z[j])
        W = W * (1 - W)
        scores.append(np.log(np.linalg.det(Prec + W * np.outer(V[j], V[j]))))
    return cand[np.argmax(scores)]

```

```
def choose_random(state, asked_mask, V, Z):
    return rng.choice(np.where(~asked_mask)[0])

print("Update and selection functions defined.")
```

Run simulation comparing D-optimal vs random selection:

```
#| autorun: true
def run_policy(policy_fn):
    states = [new_user_state() for _ in range(N)]
    asked = np.zeros((N, M), dtype=bool)
    mse_curve, rel_curve = [], []
    for t in range(T):
        for i in range(N):
            j = policy_fn(states[i], asked[i], V, Z)
            y = int(rng.random() < sigmoid(U_true[i] @ V[j] + Z[j]))
            update_user(states[i], V[j], Z[j], y)
            asked[i, j] = True
        U_hat = np.array([s["U_hat"] for s in states])
        err_tr = np.mean([np.trace(s["Sigma"]) for s in states])
        mse_curve.append(np.mean(np.sum((U_hat - U_true)**2, axis=1)))
        rel_curve.append(1 - err_tr / np.trace(np.cov(U_true.T)))
    return np.array(mse_curve), np.array(rel_curve)

mse_rand, rel_rand = run_policy(choose_random)
mse_act, rel_act = run_policy(choose_item_Dopt)
print("Simulation complete!")
```

Visualize results:

```
#| autorun: true
fig, axes = plt.subplots(1, 2)
xs = np.arange(1, T + 1)

axes[0].plot(xs, mse_rand, 'o-', label="Random", alpha=0.7)
axes[0].plot(xs, mse_act, 's-', label="D-opt", alpha=0.7)
axes[0].set_xlabel("Queries"); axes[0].set_ylabel("MSE")
axes[0].set_title("Estimation Error"); axes[0].legend(); axes[0].grid(True, alpha=0.3)

axes[1].plot(xs, rel_rand, 'o-', label="Random", alpha=0.7)
axes[1].plot(xs, rel_act, 's-', label="D-opt", alpha=0.7)
axes[1].set_xlabel("Queries"); axes[1].set_ylabel("Reliability")
axes[1].set_title("Reliability"); axes[1].legend(); axes[1].grid(True, alpha=0.3)

plt.tight_layout(); plt.show()
```

3.2.2 Pairwise Preferences

We next discuss the measurement problem with pairwise data. Recall from that the Bradley-Terry model specifies the probability that user i prefers item j over item k as $p_i(j \succ k) = \sigma(U^\top(V_j - V_k) + Z_j - Z_k)$ with feature $x_{jk} = V_j - V_k \in \mathbb{R}^K$ and offset $b_{jk} = Z_j - Z_k$.

In metric elicitation, each oracle query asks: "Which classifier is better, h_θ or $h_{\theta'}$?" The oracle's answer is deterministic: $\Omega(C_\theta, C_{\theta'}) = \mathbf{1}[\mathbf{m}^\top(V_\theta - V_{\theta'}) > 0]$, which is exactly a pairwise comparison with feature difference $x = V_\theta - V_{\theta'}$. The deterministic oracle corresponds to the $\sigma \to$ step function limit of Bradley-Terry.

For a single observation $y \in \{0, 1\}$ of $j \succ k$ the per-pair log-likelihood is

$$\ell(U; y, i, j, k) = y \log p + (1 - y) \log(1 - p_i). \tag{3.7}$$

Score and expected Fisher information for (U):

$$\nabla_U \ell = (y - p), x_{jk}, \qquad \mathcal{J}_{jk}(U) = \mathbb{E}[-\nabla_U^2 \ell] = w x_{jk} x_{jk}^\top, \tag{3.8}$$

where $w = p(1 - p)$. For a sequence of queried pairs (j_t, k_t) the cumulative information is $\mathcal{J}(U) = \sum_t w_t, x_t x_t^\top$. With a Gaussian prior $U \sim \mathcal{N}(0, \Sigma_0)$ and a Laplace approximation, the posterior precision updates additively:

$$\Lambda_t = \Sigma_t^{-1} = \Sigma_0^{-1} + \sum_{s \le t} w_s, x_s x_s^\top. \tag{3.9}$$

A single Newton step for the posterior mean after observing ((x,y)) is

$$\hat{U} \leftarrow \hat{U} + \Sigma_t (y - p), x, \qquad p = \sigma(\hat{U}^\top x + b), \quad \Sigma_t = \left(\Lambda_{t-1} + w, x x^\top\right)^{-1}. \tag{3.10}$$

This is the logistic-regression one-step IRLS update specialized to one new example. We evaluate candidates at the current $(\hat{U}, \Sigma)$. Write x for x_{jk} and $w = \sigma(\hat{U}^\top x + b)(1 - \sigma(\hat{U}^\top x + b))$. Let $\Lambda = \Sigma^{-1}$. Using the Sherman–Morrison formula, we obtain closed forms for all three design criteria:

Proposition: D-Optimal Acquisition via Sherman-Morrison

Proposition 3.2. *For a current precision matrix $\Lambda = \Sigma^{-1}$ and a candidate pair (j, k) with feature difference $x = V_j - V_k$ and variance weight $w = p(1 - p)$, the D-optimal acquisition function has closed form:*

$$\Delta_D(j, k) = \log\det(\Lambda + w x x^\top) - \log\det(\Lambda) = \log(1 + w x^\top \Sigma x)$$

This follows from the matrix determinant lemma: $\det(A + u v^\top) = (1 + v^\top A^{-1} u)\det(A)$.

Pick the pair that maximizes $\log(1 + w x^\top \Sigma x)$. This requires only a vector-matrix-vector product, making it efficient for online selection.

MYOPIC VS. NON-MYOPIC QUERY SELECTION

The D-optimal (and A/E-optimal) criteria above are **greedy**: they select each query to maximize the *immediate* information gain, without considering how the current query affects future query opportunities. The optimal policy would look ahead over a budget of T queries and select the sequence that maximizes total information. In practice, the greedy approach works well when queries are roughly exchangeable, but it can perform poorly in **batch selection** settings (selecting multiple queries simultaneously) because it ignores redundancy—two individually informative queries may provide overlapping information. Batch extensions typically add diversity mechanisms, such as penalizing queries whose feature vectors $V_j - V_k$ are correlated with already-selected pairs.

- A-optimal (trace drop of covariance): $\Delta_A(j,k); =; \mathrm{tr}(\Sigma) - \mathrm{tr}((\Lambda + w, xx^\top)^{-1}) = \frac{w, x^\top \Sigma^2 x}{1 + w, x^\top \Sigma x}$.
- E-optimal (raise the worst-direction information): Exact update of $\lambda_{\min}(\mathcal{J})$ needs an eigen update. A practical proxy is to maximize $\Delta_E^{\text{proxy}}(j,k) = w, x^\top \Sigma x$, which targets directions where current uncertainty $x^\top \Sigma x$ is large.

End-to-end toy pipeline (rank 2, known items, active D-opt vs random). First, define helper functions for D-optimal gain and Bayesian updates:

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))

def d_opt_gain(Sigma, x, w):
    """D-optimal log-det gain: log(1 + w x^T Sigma x)"""
    return np.log1p(w * x @ Sigma @ x)

def one_step_update(U_hat, Sigma, x, b, y):
    """Sherman-Morrison update for posterior mean and covariance."""
    p = sigmoid(U_hat @ x + b)
    w = p * (1 - p)
    Sx = Sigma @ x
    Sigma_new = Sigma - (w / (1.0 + w * (x @ Sx))) * np.outer(Sx, Sx)
    U_new = U_hat + Sigma_new @ ((y - p) * x)
    return U_new, Sigma_new

print("Helper functions defined.")
```

Set up simulation parameters and precompute candidate pairs:

```
#| autorun: true
rng = np.random.default_rng(0)
N, M, K, T = 200, 60, 2, 30  # users, items, rank, queries
```

```

V = rng.normal(0, 1.0, size=(M, K))  # item embeddings
Z = rng.normal(0, 0.3, size=M)       # item biases
U_true = rng.normal(0, 1.0, size=(N, K))  # true user embeddings

# Precompute all pairs and their features
pairs = np.array([(j, k) for j in range(M) for k in range(M) if j < k])
X = V[pairs[:,0]] - V[pairs[:,1]]  # pairwise feature differences
B = Z[pairs[:,0]] - Z[pairs[:,1]]  # pairwise bias differences

print(f"Setup: {N} users, {M} items, {len(pairs)} candidate pairs, {T} queries each")
```

Define selection policies and run the simulation:

```
#| autorun: true
def choose_pair_dopt(U_hat, Sigma, X, B):
    """Select pair maximizing D-optimal gain."""
    h = X @ U_hat + B
    p = sigmoid(h)
    w = p * (1 - p)
    gains = np.array([d_opt_gain(Sigma, X[i], w[i]) for i in range(len(X))])
    gains[np.isnan(gains)] = -np.inf
    return int(np.argmax(gains))

def run_policy(active=True):
    Uh = np.zeros((N, K))
    Sigmas = np.tile(np.eye(K) * 2.0, (N, 1, 1))
    mse_curve, rel_curve = [], []

    for t in range(T):
        for i in range(N):
            idx = choose_pair_dopt(Uh[i], Sigmas[i], X, B) if active else
 ↪  rng.integers(len(pairs))
            x, b = X[idx], B[idx]
            y = 1 if rng.random() < sigmoid(U_true[i] @ x + b) else 0
            Uh[i], Sigmas[i] = one_step_update(Uh[i], Sigmas[i], x, b, y)
        mse_curve.append(np.mean(np.sum((Uh - U_true)**2, axis=1)))
        rel_curve.append(1.0 - np.mean(np.trace(Sigmas, axis1=1, axis2=2)) / K)
    return np.array(mse_curve), np.array(rel_curve)

mse_d, rel_d = run_policy(active=True)
mse_r, rel_r = run_policy(active=False)
print("Simulation complete!")
```

Visualize the results:

```
#| autorun: true
fig, axes = plt.subplots(1, 2)
xs = np.arange(1, T + 1)

axes[0].plot(xs, mse_r, 'o-', label="Random", alpha=0.7)
axes[0].plot(xs, mse_d, 's-', label="D-opt", alpha=0.7)
axes[0].set_xlabel("Queries per user"); axes[0].set_ylabel("MSE")
axes[0].set_title("Estimation Error"); axes[0].legend(); axes[0].grid(True, alpha=0.3)

axes[1].plot(xs, rel_r, 'o-', label="Random", alpha=0.7)
axes[1].plot(xs, rel_d, 's-', label="D-opt", alpha=0.7)
axes[1].set_xlabel("Queries per user"); axes[1].set_ylabel("Reliability")
axes[1].set_title("Reliability"); axes[1].legend(); axes[1].grid(True, alpha=0.3)

plt.tight_layout(); plt.show()
print(f"Final MSE  | Random: {mse_r[-1]:.3f} | D-opt: {mse_d[-1]:.3f}")
print(f"Final Rel. | Random: {rel_r[-1]:.3f} | D-opt: {rel_d[-1]:.3f}")
```

Key Takeaway: D-Optimal Design

For multi-dimensional preference vectors, D-optimal selection maximizes $\log\det(\mathcal{I})$, shrinking the posterior ellipsoid in all directions. The Sherman-Morrison formula enables efficient computation: each candidate pair requires only a vector-matrix-vector product $x^\top \Sigma x$. D-optimal design naturally encourages diversity—selecting pairs with redundant information directions provides diminishing log-det gain.

3.2.3 Metric Elicitation: Exploiting Quasiconcavity

The metric elicitation problem has special structure that enables even more efficient elicitation than general D-optimal design. A *Linear Performance Metric (LPM)* has the form:

$$\phi(C) = m_{11} \cdot \text{TP} + m_{00} \cdot \text{TN} + m_0 \tag{3.11}$$

where the weights (m_{11}, m_{00}) encode tradeoffs between true positives and true negatives. Since only the ratio matters, we can parametrize as $\mathbf{m} = (\cos\theta, \sin\theta)$ for $\theta \in [0, 2\pi]$.

For medical diagnosis, $m_{11} > m_{00}$ weights sensitivity (catching disease) over specificity (avoiding false alarms). For spam filtering, the reverse may hold. The key is that the practitioner's implicit metric $\mathbf{m}^*$ is unknown—we must learn it from pairwise comparisons.

Proposition: Quasiconcavity of LPM Composition

Proposition 3.3. *For a quasiconcave LPM ϕ that is monotonically increasing in TP and TN, and a parametrization $\rho^+(\theta)$ of the upper boundary of the confusion matrix space $\mathcal{C}$, the composition $\phi \circ \rho^+$ is **quasiconcave and unimodal** on $[0, \pi/2]$.*

Quasiconcavity means the function has a single peak—no local maxima to trap us. This enables binary search rather than gradient-based optimization:

Algorithm: LPM Elicitation via Binary Search

Input: Tolerance $\epsilon > 0$, oracle Ω

Initialize: $\theta_a = 0$, $\theta_b = \pi/2$

While $|\theta_b - \theta_a| > \epsilon$:

1. Divide $[\theta_a, \theta_b]$ into four equal intervals with boundaries A, B, C, D, E
2. Compute confusion matrices C_A, C_B, C_C, C_D, C_E for classifiers at each θ
3. Query oracle: compare all adjacent pairs to find which interval contains the maximum
4. Update $[\theta_a, \theta_b]$ to the interval containing the maximum

Return: $\hat{\mathbf{m}} = (\cos\hat{\theta}, \sin\hat{\theta})$ where $\hat{\theta} = (\theta_a + \theta_b)/2$

Each iteration uses 4 queries and shrinks the search interval by factor of 2, giving $O(\log(1/\epsilon))$ **query complexity**—exponentially better than the $O(1/\epsilon^2)$ rate for general 2D estimation.

Proposition: Bayes Optimal Classifier from LPM

Proposition 3.4. *Given learned metric weights* $\mathbf{m} = (m_{11}, m_{00})$ *with* $m_{11} + m_{00} > 0$*, the Bayes optimal classifier is:*

$$\bar{h}(x) = \mathbf{1}\left[\eta(x) \geq \frac{m_{00}}{m_{11} + m_{00}}\right]$$

where $\eta(x) = P(Y = 1 \mid X = x)$ *is the conditional class probability.*

This connects metric elicitation to decision theory: learning the metric $\mathbf{m}$ is equivalent to learning the optimal classification threshold.

Extension to Linear-Fractional Metrics. The F_β score and Jaccard similarity are not linear but *linear-fractional*:

$$\phi(C) = \frac{p_{11} \cdot \mathrm{TP} + p_{00} \cdot \mathrm{TN} + p_0}{q_{11} \cdot \mathrm{TP} + q_{00} \cdot \mathrm{TN} + q_0}$$

These can be elicited by first finding both the maximizer and minimizer of the implicit linear numerator (two binary searches), then solving for the fractional parameters. The query complexity remains $O(\log(1/\epsilon))$.

Multiclass Extension. For K-class problems, diagonal linear performance metrics (DLPMs) weight correct predictions: $\psi(\mathbf{d}) = \sum_k a_k d_k$ where d_k is the rate of correctly predicting class k. Binary search over each pair of classes (k_1, k_2) recovers the relative weights a_{k_2}/a_{k_1}, with total complexity $O(K^2 \log(1/\epsilon))$.

3.3 Measurement of User Preference Function

Suppose items j have feature vectors $X_j \in \mathbb{R}^d$. Instead of learning a free scalar parameter V_j for each item, we assume $V_j = W^\top X_j$, where $W \in \mathbb{R}^d$ is shared across items. The probability

that item j beats item k is $p(j \succ k) = \sigma(V_j - V_k) = \sigma(W^\top (X_j - X_k))$. This is simply logistic regression on pairwise feature differences. Given one comparison outcome $y \in \{0, 1\}$ for pair (j, k) with difference $x_{jk} = X_j - X_k$,

$$\ell(W) = y \log \sigma(W^\top x_{jk}) + (1 - y) \log(1 - \sigma(W^\top x_{jk})). \tag{3.12}$$

Gradient: $\nabla_W \ell = (y - p) x_{jk}, \quad p = \sigma(W^\top x_{jk})$. Fisher Information (expected negative Hessian): $\mathcal{I}(W) = \mathbb{E}[(y - p)^2 x_{jk} x_{jk}^\top] = p(1 - p), x_{jk} x_{jk}^\top$.

With Gaussian prior $W \sim \mathcal{N}(0, \sigma_0^2 I)$, a Laplace approximation yields posterior covariance $\Sigma \approx (\sigma_0^{-2} I + \sum_t p_t (1 - p_t) x_t x_t^\top)^{-1}$. So, as in the Rasch case, Fisher information accumulates to form posterior precision. At each step, pick a candidate pair (j, k) and compute

$$\Delta_D(j, k) = \log \det(\Lambda + w x_{jk} x_{jk}^\top) - \log \det(\Lambda), \quad w = p(1 - p), \tag{3.13}$$

where $\Lambda = \Sigma^{-1}$. This is the D-optimal criterion, maximizing the expected log-determinant gain in information. A-optimal or E-optimal criteria are similarly defined. We'll simulate item features, a ground-truth W, and comparisons. Then we compare active D-optimal elicitation vs random pair selection for estimating W. Performance metrics are MSE of $\hat{W}$ vs true W, and reliability.

First, set up the simulation with item features and ground-truth weights:

Code

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))
rng = np.random.default_rng(0)

d = 5          # feature dimension
M = 50         # items
T = 100        # queries (pairs)
sigma0 = 2.0   # prior SD

X = rng.normal(0, 1, (M, d))
W_true = rng.normal(0, 1, d)
print(f"Setup: {M} items, {d}-dim features, {T} queries")
```

Define the Fisher gain and Bayesian update functions:

Code

```
#| autorun: true
def fisher_gain(Sigma, x, w):
    """D-optimal log-det gain."""
    Sx = Sigma @ x
    denom = 1 + w * (x @ Sx)
    Sigma_new = Sigma - (w / denom) * np.outer(Sx, Sx)
    return np.log(np.linalg.det(Sigma) / np.linalg.det(Sigma_new))

def update(W_hat, Sigma, x, y):
    """Newton update with Laplace covariance."""
    p = sigmoid(W_hat @ x)
    w = p * (1 - p)
    Sx = Sigma @ x
    denom = 1 + w * (x @ Sx)
    Sigma_new = Sigma - (w / denom) * np.outer(Sx, Sx)
    W_new = W_hat + Sigma_new @ ((y - p) * x)
    return W_new, Sigma_new

pairs = np.array([(j, k) for j in range(M) for k in range(M) if j < k])
Xdiff = X[pairs[:, 0]] - X[pairs[:, 1]]
print(f"Functions defined. {len(pairs)} candidate pairs.")
```

Run the simulation comparing D-optimal vs random selection:

Code

```
#| autorun: true
def run_policy(active=True):
    W_hat = np.zeros(d)
    Sigma = np.eye(d) * sigma0**2
    mse_hist, rel_hist = [], []

    for t in range(T):
        if active:
            gains = [fisher_gain(Sigma, x, sigmoid(W_hat @ x) * (1 -
↪   sigmoid(W_hat @ x)))
                     for x in Xdiff]
            idx = np.argmax(gains)
        else:
            idx = rng.integers(len(Xdiff))

        x = Xdiff[idx]
        y = rng.random() < sigmoid(W_true @ x)
        W_hat, Sigma = update(W_hat, Sigma, x, y)
        mse_hist.append(np.sum((W_hat - W_true)**2))
        rel_hist.append(1 - np.trace(Sigma) / (d * np.var(W_true)))
    return np.array(mse_hist), np.array(rel_hist)

mse_act, rel_act = run_policy(True)
mse_rnd, rel_rnd = run_policy(False)
print("Simulation complete!")
```

Visualize the results:

Code

```
#| autorun: true
fig, axes = plt.subplots(1, 2)

axes[0].plot(mse_rnd, label="Random", alpha=0.7)
axes[0].plot(mse_act, label="D-optimal", alpha=0.7)
axes[0].set_xlabel("Queries"); axes[0].set_ylabel("MSE")
axes[0].set_title("Estimation Error"); axes[0].legend(); axes[0].grid(True,
 ↪  alpha=0.3)

axes[1].plot(rel_rnd, label="Random", alpha=0.7)
axes[1].plot(rel_act, label="D-optimal", alpha=0.7)
axes[1].set_xlabel("Queries"); axes[1].set_ylabel("Reliability")
axes[1].set_title("Reliability"); axes[1].legend(); axes[1].grid(True, alpha=0.3)

plt.tight_layout(); plt.show()
```

3.3.1 Robotic Trajectory Learning

The linear preference framework applies directly to robotic trajectory learning (Biyik and Sadigh 2018). Consider a robot choosing between trajectories ξ_A and ξ_B. The human evaluates them via a reward function:

$$R(\xi) = w^\top \phi(\xi) \tag{3.14}$$

where $w \in \mathbb{R}^N$ are unknown weights and $\phi(\xi) \in \mathbb{R}^N$ extracts features (e.g., distance to lane boundaries, proximity to obstacles, heading, speed). This is exactly our linear model with "items" being trajectories and "features" being $\phi(\xi)$.

Geometric Interpretation. Each pairwise comparison "prefer ξ_A over ξ_B" yields a constraint:

$$w^\top(\phi(\xi_A) - \phi(\xi_B)) > 0$$

This defines a **half-space** in the weight space. The feasible region for w is the intersection of all half-spaces consistent with observed preferences. Each query splits this region—orthogonal queries reduce volume by approximately half.

The optimal query selection criterion maximizes the minimum volume removed regardless of the oracle's answer. For query feature difference $\Phi = \phi(\xi_A) - \phi(\xi_B)$, this is $\max_\Phi \min\{\mathbb{E}[1 - f_\Phi(w)], \mathbb{E}[1 - f_{-\Phi}(w)]\}$ where $f_\Phi(w) = \min(1, \exp(w^\top \Phi))$ soft-weights consistency with the preference. This selects balanced queries where either response is informative.

Driving Simulator Example. In a 2D driving simulator, trajectory features include:

- Distance to lane boundaries
- Distance to other vehicles

- Heading angle
- Speed

With no learned preferences, the simulated car moves erratically. After 30 comparison queries, it learns to follow the lane direction. After 70 queries, it additionally learns to avoid collisions while staying in lane. The features with highest learned weights correspond to collision avoidance and lane-keeping.

Why Active Selection Matters. Both active and random query selection eventually converge to the true reward weights. However, active selection achieves target performance **faster**—critical in time-sensitive applications. For exoskeleton rehabilitation (Li et al. 2021), the robot must adapt to individual patients quickly; waiting for asymptotic convergence is not acceptable. Active learning provides crucial early-stage advantages.

Imitation Learning with Compatibility. When learning from multiple human demonstrators, *multimodality* arises: different people use different strategies for the same task (e.g., picking up an object from the left vs. right). Naive imitation learning fails on contradictory demonstrations.

The solution is to measure **compatibility** of new demonstrations with the base policy:

- **Likelihood**: How close is the demonstrated action to the base policy's prediction?
- **Novelty**: How uncertain is the base policy at this state?

Demonstrations with low likelihood in low-novelty states indicate conflicting modes and should be filtered. This compatibility-based filtering improves success rates while using less data than naive aggregation (Gandhi et al. 2022).

Foundation Models for Representations. Recent work leverages foundation models pretrained on human video (R3M (Nair et al. 2022), Voltron (Karamcheti et al. 2023)) to provide better trajectory features $\phi(\xi)$. These models learn that human hands are similar to robotic end-effectors, enabling 2-3x higher success rates with 5-10x fewer demonstrations. For preference learning, better representations mean each comparison is more informative.

3.3.2 Non-Linear Scoring Functions

So far we set $V_j = W^\top X_j$. Suppose instead that items are scored by a non-linear function

$$V_j = f_\theta(X_j), \qquad \theta \in \mathbb{R}^p, \qquad p(j \succ k) = \sigma(f_\theta(X_j) - f_\theta(X_k)) \tag{3.15}$$

We want an online procedure that chooses the next comparison (j, k) to learn θ quickly. Exact Bayesian design is intractable for general f_θ. There is a simple and effective approximation that keeps all the math and code lightweight: local Gaussian posterior over θ via Laplace + first-order model. Work at the current estimate $\hat{\theta}$. Linearize the non-linear scorer around $\hat{\theta}$:

$$f_\theta(X_j) \approx f_{\hat{\theta}}(X_j) + J_j \Delta\theta, \qquad J_j \equiv \left.\frac{\partial f_\theta(X_j)}{\partial \theta}\right|_{\theta=\hat{\theta}} \in \mathbb{R}^{1\times p}. \tag{3.16}$$

The pairwise logit for (j, k) becomes

$$\underbrace{f_\theta(X_j) - f_\theta(X_k)}_{\text{logit}} \approx \underbrace{f_{\hat{\theta}}(X_j) - f_{\hat{\theta}}(X_k)}_{\text{offset}} + \underbrace{(J_j - J_k)}_{\phi_{jk}} \Delta\theta. \tag{3.17}$$

Define the local "design vector" $\phi_{jk} \in \mathbb{R}^{1\times p}$. With a Gaussian prior $\theta \sim \mathcal{N}(0, \sigma_0^2 I)$ and Bernoulli likelihood, the Laplace posterior around $\hat{\theta}$ is

$$\theta \mid \mathcal{D} \approx \mathcal{N}(\hat{\theta}, ; \Sigma), \qquad \Sigma^{-1} = \sigma_0^{-2} I + \sum_t w_t \phi_t^\top \phi_t \tag{3.18}$$

where $w_t = p_t(1 - p_t)$ with $p_t = \sigma(f_{\hat{\theta}}(X_{j_t}) - f_{\hat{\theta}}(X_{k_t}))$. This is exactly the same structure as the linear case, with (x_{jk}) replaced by $\phi_{jk} = J_j - J_k$. You only need Jacobian rows J_j. At the current $(\hat{\theta}\Sigma)$ and for a candidate (j, k), compute $p = \sigma(f_{\hat{\theta}}(X_j) - f_{\hat{\theta}}(X_k))$ and $w = p(1 - p)$.Compute $\phi = J_j - J_k$ at $\hat{\theta}$. Then use any standard optimal design score with Sherman–Morrison:

- D-optimal (log-det gain): $\Delta_D(j, k) = \log(1 + w\phi\Sigma\phi^\top)$.
- A-optimal (trace drop): $\Delta_A(j, k) = \dfrac{w\phi\Sigma^2\phi^\top}{1 + w\phi\Sigma\phi^\top}$.
- E-optimal proxy: $\Delta_E^{\text{proxy}}(j, k) = w\phi\Sigma\phi^\top$.

Pick the pair with the largest gain. This is tractable because it only needs a vector–matrix–vector product with Σ and a Jacobian row. Mutual-information acquisition like BALD reduces to the same formulas under the Laplace Gaussian approximation. Maximizing (□_D) is a good default.

We use a tiny two-layer network $f_\theta(x) = a^\top \tanh(Bx)$ with $\theta = (a, B)$. We derive J_j analytically. First, define the network and its Jacobian:

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))
rng = np.random.default_rng(3)

# Network dimensions
d, h, M, T = 6, 8, 80, 120  # feature dim, hidden units, items, queries

# Item features and ground-truth parameters
X = rng.normal(0, 1, size=(M, d))
a_star = rng.normal(0, 0.8, size=(h,))
B_star = rng.normal(0, 0.6, size=(h, d))

def f_theta(a, B, x):
```

```
    """Two-layer network: a^T tanh(Bx)"""
    return a @ np.tanh(B @ x)

def f_batch(a, B, X):
    """Batch evaluation over all items."""
    return a @ np.tanh(B @ X.T)

def J_row(a, B, x):
    """Jacobian  f/  at x, with   = [a, vec(B)]."""
    t = np.tanh(B @ x)
    sech2 = 1.0 - t**2
    dfa = t  #  f/ a
    dfB = (a[:, None] * sech2[:, None]) * x[None, :]  #  f/ B
    return np.concatenate([dfa, dfB.ravel()])

V_star = f_batch(a_star, B_star, X)  # true scores
print(f"Network: {d}→{h}→1, {h + h*d} parameters, {M} items")
```

Set up Laplace approximation state and update functions:

```
#| autorun: true
p_dim = h + h*d  # total parameters
theta_hat = np.zeros(p_dim)
Sigma = np.eye(p_dim) * 4.0

def split_theta(theta):
    return theta[:h], theta[h:].reshape(h, d)

def update(theta, Sigma, j, k, y):
    """Laplace update after observing comparison (j,k) with outcome y."""
    a, B = split_theta(theta)
    pjk = sigmoid(f_theta(a, B, X[j]) - f_theta(a, B, X[k]))
    w = pjk * (1 - pjk)
    phi = J_row(a, B, X[j]) - J_row(a, B, X[k])
    Sphi = Sigma @ phi
    Sigma_new = Sigma - (w / (1 + w * (phi @ Sphi))) * np.outer(Sphi, Sphi)
    theta_new = theta + Sigma_new @ ((y - pjk) * phi)
    return theta_new, Sigma_new

def d_opt_gain(Sigma, phi, w):
    return np.log1p(w * (phi @ (Sigma @ phi)))

pairs = np.array([(j, k) for j in range(M) for k in range(M) if j < k])
print(f"Setup complete: {len(pairs)} candidate pairs")
```

Run the active learning loop comparing D-optimal vs random selection:

```
#| autorun: true
def run(active=True):
    theta, S = theta_hat.copy(), Sigma.copy()
    mse_hist, rel_hist = [], []

    for t in range(T):
        a, B = split_theta(theta)
        if active:
            # Heuristic: evaluate on subset of pairs with small |logit|
            logits = f_batch(a, B, X)
            cand_idx = rng.choice(len(pairs), size=min(2000, len(pairs)),
 ↪  replace=False)
            deltas = logits[pairs[cand_idx, 0]] - logits[pairs[cand_idx, 1]]
            sel = cand_idx[np.argsort(np.abs(deltas))[:400]]
            gains = [d_opt_gain(S, J_row(a, B, X[j]) - J_row(a, B, X[k]),
                     sigmoid(logits[j]-logits[k])*(1-sigmoid(logits[j]-logits[k])))
                     for j, k in pairs[sel]]
            j, k = pairs[sel[np.argmax(gains)]]
        else:
            j, k = pairs[rng.integers(len(pairs))]

        y = 1 if rng.random() < sigmoid(V_star[j] - V_star[k]) else 0
        theta, S = update(theta, S, j, k, y)
        err = theta - np.concatenate([a_star, B_star.ravel()])
        mse_hist.append(float(err @ err))
        rel_hist.append(1 - np.trace(S) / (np.trace(S) + p_dim))
    return np.array(mse_hist), np.array(rel_hist)

mse_act, rel_act = run(active=True)
mse_rnd, rel_rnd = run(active=False)
print("Simulation complete!")
```

Visualize results:

```
#| autorun: true
fig, axes = plt.subplots(1, 2)
x = np.arange(1, T+1)

axes[0].plot(x, mse_rnd, 'o-', label="Random", alpha=0.7, markevery=10)
axes[0].plot(x, mse_act, 's-', label="D-opt", alpha=0.7, markevery=10)
axes[0].set_xlabel("Queries"); axes[0].set_ylabel("MSE in  ")
axes[0].set_title("Parameter Estimation Error"); axes[0].legend(); axes[0].grid(True,
 ↪  alpha=0.3)

axes[1].plot(x, rel_rnd, 'o-', label="Random", alpha=0.7, markevery=10)
axes[1].plot(x, rel_act, 's-', label="D-opt", alpha=0.7, markevery=10)
axes[1].set_xlabel("Queries"); axes[1].set_ylabel("Reliability")
```

```
axes[1].set_title("Reliability Proxy"); axes[1].legend(); axes[1].grid(True,
    alpha=0.3)

plt.tight_layout(); plt.show()
```

3.4 Active Query Selection for Gaussian Process Models

The previous sections developed Fisher information for optimal query selection in *parametric* preference models—the Rasch model (Section 1) and linear scoring functions (Section 2). However, Chapter 2 introduced Gaussian Processes (GPs) as a flexible nonparametric alternative for reward modeling, and Chapter 3 covered GP inference via Laplace approximation. Here we address the natural question: **how do we actively select queries when using GPs?**

The key insight is that GP uncertainty—quantified by the posterior variance—naturally guides query selection. Unlike parametric models where Fisher information depends only on the current parameter estimate, GP uncertainty varies across the input space, being higher far from observed data and lower near it.

3.4.1 Information-Theoretic Acquisition Function

For GP preference learning, we seek queries that maximize information gain about the reward function. Given a dataset $\mathcal{D}$ and a candidate query $Q = (x_A, x_B)$, the acquisition function measures the expected reduction in uncertainty about r:

$$a(Q) = I(r; y \mid \mathcal{D}, Q) = H(y \mid \mathcal{D}, Q) - \mathbb{E}_{r \sim p(r|\mathcal{D})}[H(y \mid r, Q)] \tag{3.19}$$

Interpretation of the two terms:

1. $H(y \mid \mathcal{D}, Q)$: Predictive entropy—how uncertain is the query outcome under our current model? High when $p(A \succ B) \approx 0.5$.
2. $\mathbb{E}_r[H(y \mid r, Q)]$: Expected conditional entropy—how uncertain would the query be if we knew the true reward function? Low for "easy" comparisons, high for inherently noisy comparisons.

The acquisition function trades off these two terms: we want queries where the *model* is uncertain (high predictive entropy) but a *human* could provide a clear answer (low expected conditional entropy).

3.4.2 Practical Computation

Using the Laplace approximation from Chapter 3 (), the acquisition function can be computed as:

$$a(x_A, x_B) = h\left(\Phi\left(\frac{\mu_A - \mu_B}{\sqrt{2\sigma^2_{\text{noise}} + g(x_A, x_B)}}\right)\right) - m(x_A, x_B) \tag{3.20}$$

where: - μ_A, μ_B are the GP posterior means at x_A, x_B - $g(x_A, x_B) = \sigma_A^2 + \sigma_B^2 - 2\text{Cov}(r(x_A), r(x_B))$ captures uncertainty from both points - $h(p) = -p\log_2 p - (1-p)\log_2(1-p)$ is binary entropy - Φ is the standard normal CDF - $m(x_A, x_B)$ is a correction term for human noise

Key property: The trivial query $Q = (x, x)$ (comparing an item to itself) is a *global minimizer*, not maximizer, of this acquisition function. This contrasts with some other methods that can get stuck on uninformative queries.

First, we define the RBF kernel and helper functions for the simulation:

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)

def rbf_kernel(X1, X2, sigma_f=1.0, length_scale=0.7):
    """Compute RBF kernel matrix between two sets of points."""
    sq_dist = np.sum(X1**2, 1, keepdims=True) + np.sum(X2**2, 1) - 2*X1@X2.T
    return sigma_f**2 * np.exp(-sq_dist / (2 * length_scale**2))

sigmoid = lambda x: 1.0 / (1.0 + np.exp(-np.clip(x, -500, 500)))
binary_entropy = lambda p: -p*np.log2(p+1e-10) - (1-p)*np.log2(1-p+1e-10)

# True reward function (unknown to the learner)
def true_reward(x):
    return np.sin(2 * x) + 0.3 * x

# Visualize the true reward
X_grid = np.linspace(-3, 3, 100).reshape(-1, 1)
plt.figure()
plt.plot(X_grid, true_reward(X_grid), 'k-', lw=2)
plt.xlabel('x'); plt.ylabel('r(x)')
plt.title('True Reward Function (Unknown to Learner)')
plt.grid(True, alpha=0.3)
plt.show()
```

Next, we implement a GP model for active preference learning. The `fit` method uses Laplace approximation (Newton's method) to find the posterior mode, and `acquisition` computes the information gain for a candidate query:

```
#| autorun: true
class GPActivePreference:
    def __init__(self, X_pool, sigma_noise=0.1):
        self.X_pool = X_pool
        self.sigma_noise = sigma_noise
        self.comparisons = []  # list of (i, j, y) where y=1 means i>j
        self.r = np.zeros(len(X_pool))
        self.Sigma = rbf_kernel(X_pool, X_pool) + 1e-4*np.eye(len(X_pool))

    def fit(self):
        """Fit GP using Laplace approximation (Newton's method)."""
        if len(self.comparisons) == 0:
            return
        m = len(self.X_pool)
        self.K = rbf_kernel(self.X_pool, self.X_pool) + 1e-4*np.eye(m)
        self.K_inv = np.linalg.inv(self.K)

        # Newton's method for mode finding
        self.r = np.zeros(m)
        for _ in range(10):
            grad, W = np.zeros(m), np.zeros(m)
            for i, j, y in self.comparisons:
                p = sigmoid(self.r[i] - self.r[j])
                grad[i] += (y - p); grad[j] -= (y - p)
                W[i] += p * (1 - p); W[j] += p * (1 - p)
            H = -np.diag(W + 1e-6) - self.K_inv
            self.r -= np.linalg.solve(H, grad - self.K_inv @ self.r)
        self.W = W
        self.Sigma = np.linalg.inv(self.K_inv + np.diag(W + 1e-6))

    def acquisition(self, i, j):
        """Information gain acquisition: entropy of predicted comparison."""
        if len(self.comparisons) == 0:
            return 0.5
        var_diff = self.Sigma[i,i] + self.Sigma[j,j] - 2*self.Sigma[i,j]
        pred_std = np.sqrt(2*self.sigma_noise**2 + var_diff)
        p_pred = sigmoid((self.r[i] - self.r[j]) / pred_std)
        return binary_entropy(p_pred)

    def select_query(self):
        """Select the query with maximum information gain."""
        m = len(self.X_pool)
        best_score, best_pair = -np.inf, (0, 1)
        for i in range(m):
            for j in range(i+1, m):
                score = self.acquisition(i, j)
                if score > best_score:
                    best_score, best_pair = score, (i, j)
        return best_pair
```

```
    def add_comparison(self, i, j):
        """Simulate oracle response and update model."""
        r_true = true_reward(self.X_pool.flatten())
        p = sigmoid(r_true[i] - r_true[j])
        y = 1 if np.random.rand() < p else 0
        self.comparisons.append((i, j, y))
        self.fit()

print("GPActivePreference class defined successfully.")
```

Now we run the simulation, comparing active (information-theoretic) query selection against random selection:

```
#| autorun: true
# Setup
X_pool = np.linspace(-3, 3, 30).reshape(-1, 1)
n_queries = 25
r_true = true_reward(X_pool.flatten())

# Active selection
gp_active = GPActivePreference(X_pool)
mse_active = []

# Random selection
gp_random = GPActivePreference(X_pool)
mse_random = []

# Run comparison
for t in range(n_queries):
    # Active: select most informative query
    i, j = gp_active.select_query()
    gp_active.add_comparison(i, j)
    mse_active.append(np.mean((gp_active.r - r_true)**2))

    # Random: select query uniformly at random
    i, j = np.random.choice(len(X_pool), 2, replace=False)
    gp_random.add_comparison(i, j)
    mse_random.append(np.mean((gp_random.r - r_true)**2))

print(f"Simulation complete: {n_queries} queries collected for each method.")
```

Finally, we visualize the results. The left panel shows how estimation error decreases with more comparisons; the right panel shows the learned reward functions:

```
#| autorun: true
fig, axes = plt.subplots(1, 2)
```

```

# Learning curves
axes[0].plot(range(1, n_queries+1), mse_random, 'o-', label='Random', alpha=0.7)
axes[0].plot(range(1, n_queries+1), mse_active, 's-', label='Active (Info Gain)',
    alpha=0.7)
axes[0].set_xlabel('Number of comparisons')
axes[0].set_ylabel('MSE of reward estimate')
axes[0].set_title('GP Active vs Random Query Selection')
axes[0].legend(); axes[0].grid(True, alpha=0.3)

# Learned rewards
axes[1].plot(X_pool, r_true, 'k--', lw=2, label='True reward')
axes[1].plot(X_pool, gp_active.r, 'b-', lw=2, label='Active GP')
axes[1].plot(X_pool, gp_random.r, 'r-', lw=2, alpha=0.7, label='Random GP')
axes[1].fill_between(X_pool.flatten(),
                     gp_active.r - 2*np.sqrt(np.diag(gp_active.Sigma)),
                     gp_active.r + 2*np.sqrt(np.diag(gp_active.Sigma)),
                     alpha=0.2, color='blue', label='±2 (Active)')
axes[1].set_xlabel('x'); axes[1].set_ylabel('r(x)')
axes[1].set_title(f'Learned Rewards after {n_queries} Comparisons')
axes[1].legend(); axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Final MSE - Active: {mse_active[-1]:.4f}, Random: {mse_random[-1]:.4f}")
print(f"Active achieves {mse_random[-1]/mse_active[-1]:.1f}x lower error")
```

3.4.3 When to Use GP Active Learning

Scenario	Recommendation
Nonlinear reward structure	GP strongly preferred
Low-to-moderate dimensions ($d \lesssim 10$)	GP works well
Expensive human feedback	Active learning essential
Need uncertainty quantification	GP provides this naturally
Large datasets ($n > 1000$)	Consider sparse GP approximations
Very high dimensions	Linear models may be more practical

3.5 Active Learning for Direct Preference Optimization

We now turn to a specific, important application: **active learning for DPO (Direct Preference Optimization)** in large language model alignment. Chapter 2 introduced DPO and its connection to Bradley-Terry; here we show how to apply active learning principles to reduce the number of human comparisons needed.

3.5.1 The DPO Setting

Recall from that DPO directly optimizes a policy π_θ on preference data without learning an explicit reward model. The key insight is that DPO implicitly assumes Bradley-Terry preferences:

$$p(y_w \succ y_l \mid x) = \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right) \tag{3.21}$$

Challenge: Large language models have billions of parameters, making exact Fisher information computation intractable. How can we apply the D-optimal design principles from Sections 1-2?

3.5.2 The Log-Linear Policy Assumption

The key insight is to approximate the policy as **log-linear** in the last-layer features:

$$\pi(y \mid x; \theta) \propto \exp(\phi(x, y)^\top \theta) \tag{3.22}$$

where $\phi(x, y)$ is the feature representation from the model's last layer. This assumption is justified by the **neural tangent kernel** perspective: for sufficiently large networks, the dynamics during training are approximately linear in the parameters.

Under this assumption, the DPO loss Hessian takes the form:

$$H(\theta) = \sum_{i=1}^{n} p_i(1 - p_i) \cdot \Delta\phi_i \Delta\phi_i^\top \tag{3.23}$$

where $\Delta\phi_i = \phi(x_i, y_{w,i}) - \phi(x_i, y_{l,i})$ is the feature difference between winning and losing responses, and p_i is the predicted preference probability.

Key observation: This Hessian is exactly the **Fisher information matrix** for the DPO model! The D-optimal criterion becomes:

$$\max_S \log\det\left(\sum_{i \in S} p_i(1 - p_i) \cdot \Delta\phi_i \Delta\phi_i^\top\right) \tag{3.24}$$

3.5.3 The ADPO Algorithm

The **Active DPO (ADPO)** algorithm applies D-optimal design to select preference comparisons:

Algorithm: Active DPO (ADPO)

Input: Reference policy π_{ref}, prompt set $\{x_i\}$, response pairs $\{(y_{i,1}, y_{i,2})\}$, budget T

Initialize: $\theta_0 = \theta_{\text{ref}}$, $S_0 = \emptyset$

For $t = 1, \ldots, T$:

1. **Compute features**: For each candidate pair $(x_i, y_{i,1}, y_{i,2})$:
   - $\Delta\phi_i = \phi(x_i, y_{i,1}) - \phi(x_i, y_{i,2})$
   - $p_i = \sigma(\beta(\log \pi_{\theta_{t-1}}(y_{i,1}|x_i) - \log \pi_{\theta_{t-1}}(y_{i,2}|x_i)))$
2. **D-optimal selection**: Select query maximizing:

$$i^* = \arg\max_i \log\det\left(I + p_i(1-p_i)H_t^{-1}\Delta\phi_i\Delta\phi_i^\top\right) \tag{3.25}$$

   where H_t is the current Fisher information matrix
3. **Query oracle**: Obtain human preference y_t for pair $(x_{i^*}, y_{i^*,1}, y_{i^*,2})$
4. **Update**: $S_t = S_{t-1} \cup \{(i^*, y_t)\}$, retrain θ_t on S_t using DPO loss

Return: Final policy π_{θ_T}

When the Log-Linear Policy Assumption Fails

ADPO's theoretical guarantees rely on the **log-linear policy assumption**: that $\log \pi_\theta(y \mid x)$ is approximately linear in θ, which holds in the neural tangent kernel (NTK) regime near the reference policy. This approximation degrades when: (1) the policy is fine-tuned **far from the reference** (large $\|\theta - \theta_{\text{ref}}\|$), (2) the model is **small** relative to the data complexity so the NTK regime does not hold, or (3) the loss landscape is **highly non-convex** with multiple basins. In these settings, the Fisher information computed under the log-linear assumption may poorly reflect the actual curvature of the loss, leading to suboptimal query selection. Practitioners should monitor whether the D-optimal gains predicted by the linear model correlate with actual improvements in policy quality.

3.5.4 Theoretical Guarantees

Under standard regularity conditions, ADPO achieves the following convergence rate:

Theorem: ADPO Convergence Rate

Theorem 3.1. *Let θ^* be the optimal policy parameters. Under the log-linear policy assumption and standard regularity conditions, ADPO with n actively selected comparisons achieves:*

$$\|\hat{\theta}_n - \theta^*\| = O\left(\frac{d}{\sqrt{n}}\right) \tag{3.26}$$

where d is the effective dimension (rank of the Fisher information matrix). This matches the ***minimax optimal rate*** *for d-dimensional estimation.*

The key insight is that D-optimal design ensures the Fisher information grows proportionally to n, leading to $O(1/\sqrt{n})$ estimation error.

3.5.5 ADPO+ for Offline Settings

In many practical settings, we have access to a fixed pool of unlabeled preference pairs (e.g., model outputs) but cannot generate new queries. **ADPO+** adapts the algorithm for this offline setting:

1. **Pool-based selection**: Instead of synthesizing queries, select from the existing pool
2. **Batch selection**: Select b queries at once to enable parallel annotation
3. **Diversity regularization**: Add penalty for selecting similar queries

The D-optimal criterion naturally encourages diversity: selecting redundant queries provides diminishing information gain.

3.5.6 Empirical Guidance: When Does Active Learning Help?

Factor	Active Learning Beneficial?
Annotation cost	High cost → Active learning essential
Data heterogeneity	Diverse prompts → More room for strategic selection
Budget constraints	Limited budget → Active learning shows larger gains
Model capacity	Larger models → Active learning helps reduce overfitting
Query pool size	Large pool → More room for intelligent selection

Practical recommendation: Active learning for DPO provides the largest gains when: - Annotation budget is limited (< 10K comparisons) - Prompts cover diverse topics or difficulty levels - The goal is sample efficiency over raw performance

Comparison: GP Active Learning vs. ADPO

Aspect	GP Active Learning	ADPO
Model	Nonparametric (GP)	Parametric (neural network)
Scalability	$O(n^3)$	$O(d^2 n)$ with approximations

Use case	Small datasets, need uncertainty	LLM alignment, large models
Acquisition	Information gain	D-optimal design
Theoretical basis	Mutual information	Fisher information

Both approaches share the core insight: **use model uncertainty to guide query selection**.

3.6 Summary

- When human feedback is expensive, **which comparisons to request** matters as much as how to learn from them. This chapter develops the theory of optimal query selection for preference learning.
- **Fisher information** $\mathcal{I}_j(U) = p(1-p)$ quantifies how much a single comparison reveals about user preferences. Items with predicted win probability $p \approx 0.5$ are most informative—close competitions reveal more than blowouts.
- Three **optimal design criteria** guide query selection: **A-optimal** (minimize average variance), **D-optimal** (maximize the determinant of the information matrix, shrinking the posterior ellipsoid's volume), and **E-optimal** (minimize the worst-case variance).
- For multi-dimensional preferences, the **Sherman-Morrison update** enables efficient $O(K^2)$ covariance updates after each query, making real-time active selection practical.
- **Gaussian Process active learning** extends these ideas to nonparametric models using information-theoretic acquisition functions—selecting queries that maximize mutual information between the response and the latent preference function.
- **Active DPO (ADPO)** applies D-optimal design to language model alignment, using a log-linear approximation to make Fisher information computation tractable for billion-parameter models. ADPO achieves the minimax-optimal convergence rate $O(d/\sqrt{n})$.
- Across all settings—scalar preferences, factor models, GPs, and LLMs—the core insight is the same: **use model uncertainty to guide query selection**, focusing annotation effort where it reduces uncertainty most.

3.7 Quick Check

Test your understanding before proceeding to the exercises.

1. An item has predicted success probability $p = 0.95$. What is its Fisher information $\mathcal{I} = p(1-p)$? Would you select this item for active learning? Why or why not?

2. A-optimal design minimizes the average posterior variance; D-optimal minimizes the volume of the confidence ellipsoid. In what situation would these select different items?

3. In ADPO, why is the log-linear policy assumption ($\pi(y|x;\theta) \propto \exp(\phi(x,y)^\top\theta)$) important? What would happen without it?

Essential Reading

If you read only five papers from this chapter's references, make them these:

1. **Rasch (1960)** — The Rasch model: foundational psychometric model connecting item difficulty to person ability
2. **Fedorov (1972)** — The theory of optimal experimental design (A/D/E-optimality)
3. **Chu and Ghahramani (2005)** — Gaussian process preference learning: nonparametric Bayesian approach to utility functions
4. **Houlsby et al. (2011)** — Bayesian active learning by disagreement (BALD), connecting information gain to query selection
5. **Christiano et al. (2017)** — Deep RL from human preferences: the original active elicitation pipeline for reward learning

3.8 Discussion Questions

1. How does the reliability metric defined in this chapter relate to classical test theory in psychometrics? What assumptions underlie this correspondence?
2. Why does D-optimal design maximize the log-determinant of the information matrix rather than the trace? In what situations might A-optimal (trace-based) selection be preferable?
3. The local linearization approach for non-linear scoring functions trades off accuracy for tractability. What are the conditions under which this approximation is most accurate? When might it fail?
4. Active preference learning assumes that items' appeals are known or pre-calibrated. How might estimation errors in item parameters affect the efficiency of active selection strategies?
5. The metric elicitation framework assumes an oracle with a consistent underlying preference. How would you modify the approach to handle an oracle whose preferences evolve over time or exhibit inconsistencies?
6. In the robotics case studies, pairwise comparisons are used to learn reward functions. What are the advantages and disadvantages of pairwise comparisons versus direct reward labeling for robotic learning?
7. Foundation models like R3M and Voltron leverage pretraining on human video data. What types of robotic tasks might benefit most from this approach? What tasks might still require substantial robot-specific data?
8. The active elicitation framework for imitation learning identifies "compatible" demonstrations. How might this concept of compatibility be extended to multi-agent settings where different agents have legitimately different optimal behaviors?

3.9 Further Reading

For readers who want to go deeper into the topics introduced in this chapter, we recommend the following:

- **Fedorov (1972)**, *Theory of Optimal Experiments* — The classical reference on A/D/E-optimal experimental design, providing the mathematical foundations for the Fisher information-based query selection methods used throughout this chapter.
- **Chaloner and Verdinelli (1995)**, "Bayesian Experimental Design: A Review" — A unified information-theoretic approach to experimental design that extends the Fisher information framework to Bayesian settings, including mutual information-based criteria that are robust to model misspecification.
- **Settles (2012)**, *Active Learning* — A comprehensive survey of active learning, covering pool-based, stream-based, and query-synthesis approaches. Places the preference elicitation methods in this chapter within the broader active learning literature.
- **Linden and Glas (2010)**, *Elements of Adaptive Testing* — How Fisher information drives modern standardized testing (GRE, GMAT), providing the practical context for the Rasch model-based elicitation methods discussed in this chapter.
- **Houlsby et al. (2011)**, "Bayesian Active Learning for Classification and Preference Learning" — Introduces the BALD (Bayesian Active Learning by Disagreement) acquisition function, an information-theoretic alternative to D-optimal design that works well with GP models.
- **Sadigh et al. (2017)**, "Active Preference-Based Learning of Reward Functions" — Active query selection for robot trajectory preferences, demonstrating how the Fisher information framework applies to continuous, high-dimensional preference learning in robotics.
- **Hiranandani et al. (2019)**, "Multiclass Performance Metric Elicitation" — The paper behind the metric elicitation case study, showing how active preference queries can recover a practitioner's implicit loss function for classifier evaluation.

3.10 Exercises

We place (*), (**), and (***) for exercises we deem relatively easy, medium, and hard.

3.10.1 Fisher Information Maximum (*)

Show that for the Rasch model with $p = \sigma(U + V_j)$, the Fisher information $\mathcal{I}_j(U) = p(1-p)$ is maximized when $p = 0.5$. What does this imply about the optimal difficulty of items for measuring a user's ability?

3.10.2 Sherman-Morrison Update (**)

Derive the Sherman-Morrison formula for efficiently updating the posterior covariance when adding a single rank-1 observation. Specifically, show that if $\Sigma^{-1} \leftarrow \Sigma^{-1} + w \cdot vv^\top$, then $\Sigma \leftarrow \Sigma - \frac{w \cdot \Sigma v v^\top \Sigma}{1 + w \cdot v^\top \Sigma v}$.

3.10.3 D-Optimal vs A-Optimal (**)

Construct a 2D example where D-optimal and A-optimal criteria select different items for the next query. Interpret the difference geometrically in terms of the posterior ellipsoid.

3.10.4 Pairwise Fisher Information (**)

For the Bradley-Terry model $p(j \succ k|U) = \sigma(U^\top(V_j - V_k))$, derive the Fisher information matrix for U given a single pairwise comparison. How does this relate to the single-item case?

3.10.5 Active Selection Implementation (***)

Extend the D-optimal pairwise selection simulation to include all three optimality criteria (A, D, E). Compare their sample efficiency on a synthetic dataset of 50 items with 3-dimensional user preferences. Which criterion performs best under different prior uncertainty structures?

3.10.6 Noisy Oracle Elicitation (***)

Extend the LPM elicitation algorithm to handle noisy oracle responses where the oracle flips its answer with probability $\epsilon < 0.5$. How does the query complexity scale with ϵ?

3.10.7 Local Linearization Analysis (***)

Consider a scoring function $f_\theta(x) = a^\top \tanh(Bx)$ with $\theta = (a, B)$. Analytically derive the Jacobian $J_j = \partial f_\theta(X_j)/\partial\theta$. Under what conditions on the hidden activations $\tanh(BX_j)$ is the local Gaussian approximation most accurate?

3.10.8 Compatibility Metric Design (**)

In the active elicitation framework for imitation learning, the compatibility metric depends on thresholds λ and η. Design an algorithm that automatically selects these thresholds from a held-out validation set of demonstrations. What objective should the algorithm optimize?

References

Biyik, Erdem, and Dorsa Sadigh. 2018. "Batch Active Preference-Based Learning of Reward Functions." In *Proceedings of the 2nd Conference on Robot Learning*, edited by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, 87:519–28. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v87/biyik18a.html.

Chaloner, Kathryn, and Isabella Verdinelli. 1995. "Bayesian Experimental Design: A Review." *Statistical Science* 10 (3): 273–304.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. "Deep Reinforce-

ment Learning from Human Preferences." *Advances in Neural Information Processing Systems* 30.

Chu, Wei, and Zoubin Ghahramani. 2005. "Preference Learning with Gaussian Processes." *Proceedings of the 22nd International Conference on Machine Learning*, 137–44.

Fedorov, Valerii V. 1972. *Theory of Optimal Experiments*. Academic Press.

Gandhi, Kanishk, Siddharth Karamcheti, Madeline Liao, and Dorsa Sadigh. 2022. "Eliciting Compatible Demonstrations for Multi-Human Imitation Learning." In *Proceedings of the 6th Conference on Robot Learning (CoRL)*.

Hiranandani, Gaurush, Shant Boodaghians, Ruta Mehta, and Oluwasanmi Koyejo. 2019. "Performance Metric Elicitation from Pairwise Classifier Comparisons." In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, edited by Kamalika Chaudhuri and Masashi Sugiyama, 89:371–79. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v89/hiranandani19a.html.

Houlsby, Neil, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. 2011. "Bayesian Active Learning for Classification and Preference Learning." In *arXiv Preprint arXiv:1112.5745*.

Karamcheti, Siddharth et al. 2023. "Language-Driven Representation Learning for Robotics." *arXiv Preprint arXiv:2302.12766*.

Li, Kejun, Maegan Tucker, Erdem Biyik, Ellen Novoseller, Joel W. Burdick, Yanan Sui, Dorsa Sadigh, Yisong Yue, and Aaron D. Ames. 2021. "ROIAL: Region of Interest Active Learning for Characterizing Exoskeleton Gait Preference Landscapes." In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. https://doi.org/10.1109/icra48506.2021.9560840.

Linden, Wim J. van der, and Cees A. W. Glas. 2010. *Elements of Adaptive Testing*. New York: Springer.

Nair, Suraj, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. 2022. "R3M: A Universal Visual Representation for Robot Manipulation." https://arxiv.org/abs/2203.12601.

Rasch, Georg. 1960. *Probabilistic Models for Some Intelligence and Attainment Tests*. Danish Institute for Educational Research.

Sadigh, Dorsa, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. 2017. "Active Preference-Based Learning of Reward Functions." In *Robotics: Science and Systems*.

Settles, Burr. 2012. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

4 Decisions

Intended Learning Outcomes

By the end of this chapter you will be able to:

- **Distinguish** the measurement perspective (reducing uncertainty) from the reward maximization perspective (managing uncertainty) in preference learning.
- **Derive** Thompson Sampling for linear objectives using Laplace approximation to handle intractable posteriors.
- **Extend** Thompson Sampling to nonlinear objectives using Gaussian Processes and GP classification.
- **Define** regret (strong, weak, and Bayesian) in dueling bandit settings and explain when each notion is appropriate.
- **Compare** winner concepts—Condorcet, Borda, and Von Neumann winners—and identify when they may diverge.
- **Apply** acquisition functions (Uniform, Epsilon-Greedy, UCB, Pure Exploration, qEUBO) for sequential decision-making with preference feedback.
- **Analyze** regret bounds for Preferential Bayesian Optimization algorithms and understand their convergence guarantees.
- **Implement** Thompson Sampling and dueling bandit algorithms in practical settings.
- **(Optional) Derive** the policy gradient theorem and explain how REINFORCE and GRPO use it for preference-based optimization.

Suggested Lecture Plan

This chapter can be covered in four 50-minute lectures:

Lecture 1 (Sections 4.1–4.2): Thompson Sampling

- Motivation: From measurement to decision-making (10 min)
- Exploration-exploitation tradeoff (10 min)
- Thompson Sampling with Laplace approximation (linear case) (15 min)
- Code demo: Linear Thompson Sampling simulation (15 min)

Lecture 2 (Sections 4.2–4.3): GP Thompson Sampling and Dueling Bandits

- Gaussian Process Thompson Sampling (nonlinear case) (20 min)
- From multi-armed bandits to dueling bandits (15 min)
- Regret definitions and winner concepts (15 min)

Lecture 3 (Sections 4.3–4.4): Dueling Bandit Algorithms and Applications

- Acquisition functions: Uniform, □-Greedy, UCB (10 min)

– Interleaved Filter algorithm (15 min)
– DBGD, Sparring EXP4, Feel-good Thompson Sampling (10 min)
– Case studies: LLM response selection and clinical trials (15 min)

Lecture 4 (Sections 4.5 and 4.7): Preferential Bayesian Optimization and CIRL

– PBO problem formulation and Copeland scores (15 min)
– Acquisition functions: qEUBO, Pure Exploration, POP-BO (15 min)
– CIRL: From passive oracles to active human agents (15 min)
– Discussion: Choice architecture and responsible deployment (5 min)

Lecture 5 (Optional) (Section 4.6): RL Foundations for Preference Optimization

– MDP formulation and the LLM generation MDP (10 min)
– Policy gradient theorem derivation (15 min)
– REINFORCE, baselines, and variance reduction (10 min)
– GRPO: From RLHF to group-relative optimization (15 min)

4.1 Chapter Overview

Chapter 1 developed models for how preferences arise from latent utilities. Chapter 2 showed how to estimate these utilities from observed data. Chapter 3 addressed how to *collect* data efficiently through active query selection. This chapter tackles the complementary question: **given uncertainty about user preferences, how should we choose actions to maximize reward?**

The shift is subtle but important. Chapter 3's measurement perspective treats uncertainty as something to be *reduced*—the goal is to learn the preference function as accurately as possible. This chapter's reward maximization perspective treats uncertainty as something to be *managed*—the goal is to make good decisions *now*, even with incomplete knowledge. Consider an LLM serving system: it must decide which response to show each user, balancing exploitation (showing responses it believes are good) against exploration (trying novel responses to learn what users prefer).

Chapter Structure. We build from simple to complex settings:

1. **Thompson Sampling** (): Linear and nonlinear (GP) approaches to sequential decision-making with preference feedback
2. **Dueling Bandits** (): When actions can only be compared pairwise, introducing regret notions and winner concepts (Condorcet, Borda, von Neumann)
3. **Applications**: Case studies connecting these methods to LLM alignment and clinical trials
4. **Preferential Bayesian Optimization**: Advanced acquisition functions (qEUBO, POP-BO) with convergence guarantees
5. **RL Foundations (Optional)** (): MDP formulation, the policy gradient theorem, and GRPO for preference optimization
6. **Human-Agent Cooperation (CIRL)**: Moving beyond passive oracles to model humans as strategic agents

Connections to Prior Chapters:

Concept	Chapters 1–3	This Chapter
Bradley-Terry model	Likelihood, MLE, Fisher info	Dueling bandit preference model, LLM selection
Laplace approximation	GP posterior inference	Thompson Sampling posterior
Acquisition functions	D-optimal, info gain (Ch. 3)	UCB, qEUBO, Pure Exploration
DPO / reward learning	Objective function, ADPO (Ch. 3)	Reward maximization under uncertainty
Policy gradient / GRPO	RLHF pipeline (Ch. 1)	Full derivation, GRPO algorithm (Optional)
Winner concepts	—	Condorcet, Borda, von Neumann → reappear in Ch. 5

4.2 The Reward Maximization under Uncertainty Problem

Historical Note: Thompson Sampling

Thompson Sampling dates back to 1933, when William Thompson proposed it for clinical trials. For decades it was largely forgotten, overshadowed by frequentist methods like UCB (Upper Confidence Bound). Its rediscovery in the 2010s, particularly for online advertising and recommendation systems, revealed that this simple Bayesian algorithm often outperforms more complex alternatives. The key insight: **sample a plausible world from your beliefs, then act optimally within it.**

The fundamental tension in sequential decision-making is the **exploration-exploitation trade-off**:

- **Exploitation**: Choose actions believed to be good based on current knowledge
- **Exploration**: Try uncertain actions to gain information for better future decisions

A purely exploitative strategy may miss superior options it never tried. A purely explorative strategy wastes resources on suboptimal choices. Effective algorithms balance both.

Assumption: Stationary Utilities

The bandit framework assumes item utilities are **fixed over time**: the reward distribution for each arm does not change across rounds. All regret bounds in this chapter rely on this stationarity assumption. In practice, preferences drift—users' tastes evolve, LLM responses improve through training, and the item set itself may change (new products, new model versions). When utilities are non-stationary, standard regret bounds no longer apply, and algorithms must explicitly account for drift, for instance by discounting old observations (as the Elo K-factor does implicitly in Chapter 2) or by using non-stationary bandit algorithms with sliding windows or change-point detection.

Each user i interacts with a set of available items indexed by j, each characterized by a known parameter vector $V_j \in \mathbb{R}^K$. The user's utility function is $H_{ij} = h_i(V_j)$, where $h_i : \mathbb{R}^K \to \mathbb{R}$ maps item features to latent utility. We only observe noisy or partial feedback, such as item-wise or pairwise responses. The agent's objective is to select items sequentially to maximize reward.

In the **linear case**, we assume $h_i(V_j) = U_i^\top V_j$, with $U_i \in \mathbb{R}^K$ representing the user's latent vector. This recovers the logistic or Rasch model when observations are binary: $p(Y_{ij} = 1) = \sigma(U_i^\top V_j)$. In this setting, the uncertainty about U_i defines a posterior distribution $p(U_i \mid \text{data})$, and the system's exploration–exploitation tradeoff can be quantified by the expected utility and posterior variance. Algorithms such as Thompson Sampling operate directly on this posterior, balancing exploration (reducing uncertainty about U_i) and exploitation (recommending high-utility V_j).

In the **nonlinear case**, a common choice is to place a Gaussian process prior over h_i: $h_i \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$, where the kernel $k(V, V')$ encodes similarity between items. This yields the Bayesian optimization formulation. The system uses acquisition functions (e.g., Thompson Sampling or expected improvement) to trade off exploitation and exploration.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

Code: Exploration vs Exploitation Visualization

```
#| autorun: true
# Visualize the exploration-exploitation tradeoff
np.random.seed(42)

# True reward function (unknown to agent)
n_arms = 10
true_means = np.random.beta(2, 5, n_arms) * 0.8 + 0.1  # Random rewards in [0.1, 0.9]
best_arm = np.argmax(true_means)

def simulate_policy(policy_fn, T=200):
    """Simulate a policy and track cumulative reward."""
    rewards = []
    arm_counts = np.zeros(n_arms)
    arm_rewards = np.zeros(n_arms)

    for t in range(T):
        arm = policy_fn(t, arm_counts, arm_rewards)
        reward = np.random.binomial(1, true_means[arm])
        arm_counts[arm] += 1
        arm_rewards[arm] += reward
        rewards.append(reward)

    return np.cumsum(rewards), arm_counts

# Greedy (pure exploitation after initial exploration)
def greedy(t, counts, rewards):
    if t < n_arms:
        return t
    means = np.where(counts > 0, rewards / counts, 0)
    return np.argmax(means)

# Random (pure exploration)
def random_policy(t, counts, rewards):
    return np.random.randint(n_arms)

# Epsilon-greedy (balanced)
def eps_greedy(t, counts, rewards, eps=0.1):
    if np.random.rand() < eps or t < n_arms:
        return np.random.randint(n_arms)
    means = np.where(counts > 0, rewards / counts, 0)
    return np.argmax(means)

# Thompson Sampling (Bayesian)
def thompson(t, counts, rewards):
    samples = np.random.beta(rewards + 1, counts - rewards + 1)
    return np.argmax(samples)

# Run simulations
T = 200
cum_greedy, _ = simulate_policy(greedy, T)
cum_random, _ = simulate_policy(random_policy, T)
cum_eps, _ = simulate_policy(lambda t, c, r: eps_greedy(t, c, r, 0.1), T)
```

4.3 Thompson Sampling under Linear Objective

Thompson Sampling (TS) is among the most conceptually transparent and empirically successful methods for decision-making under uncertainty. It converts Bayesian inference into a simple randomized decision rule: sample a plausible world, then act optimally within it. Formally, suppose we maintain a posterior distribution over the latent user embedding $p(U_i \mid \mathcal{D}_t) \propto p(\mathcal{D}_t \mid U_i)p(U_i)$, where $\mathcal{D}_t = (V_j, Y_{ij})_{j=1}^{t}$ denotes the user's observed responses. At each round $t+1$, Thompson Sampling draws a single sample $\tilde{U}_i^{(t)} \sim p(U_i \mid \mathcal{D}_t)$, and then chooses the next item that maximizes the sampled expected utility: $j^* = \arg\max_j \tilde{U}_i^{(t)\top} V_j$. This randomized strategy naturally balances exploration and exploitation: if posterior uncertainty is large, different draws $\tilde{U}_i^{(t)}$ lead to diverse item choices; as the posterior concentrates, the policy converges to greedy exploitation.

Common Pitfall: Thompson Sampling Requires a Correct Posterior

Thompson Sampling's theoretical guarantees depend on maintaining an **accurate posterior** over the latent parameters. A common mistake is to use a poor approximation (e.g., a point estimate, or a Gaussian approximation when the true posterior is multimodal) and still expect TS to explore effectively. If the posterior is overconfident, TS degenerates to greedy exploitation too early; if underconfident, it wastes rounds exploring suboptimal items. The **Laplace approximation** works well for logistic likelihoods (Bradley-Terry), but always check that the posterior is unimodal. Also, do **not** add explicit exploration bonuses (as in UCB) on top of Thompson Sampling — the randomization *is* the exploration mechanism, and adding bonuses breaks the balance.

Worked Example: Thompson Sampling Intuition

Suppose we have 3 items and a 1D user embedding. After 5 observations, the posterior over U is approximately $\mathcal{N}(1.2, 0.5^2)$.

Round 1: Draw $\tilde{U} = 1.6$ (high sample). With item features $V_1 = 0.5, V_2 = 1.0, V_3 = -0.3$:

- Expected utilities: $\tilde{U} \cdot V_1 = 0.80$, $\tilde{U} \cdot V_2 = 1.60$, $\tilde{U} \cdot V_3 = -0.48$
- Select $j^* = 2$ (highest utility under this sample)

Round 2: Draw $\tilde{U} = 0.7$ (low sample). Now:

- Expected utilities: 0.35, 0.70, -0.21
- Still select $j^* = 2$, but the margin is smaller

Round 3: Draw $\tilde{U} = -0.1$ (rare tail sample). Now:

- Expected utilities: -0.05, -0.10, 0.03
- Select $j^* = 3$ — **exploration!** The unlikely sample led us to try a different item.

Key insight: Thompson Sampling explores *automatically* because posterior uncertainty generates diverse samples. Early on (wide posterior), many different items get selected. As the posterior narrows around the true U, exploration decreases naturally — no explicit exploration schedule needed.

Under binary or preference feedback modeled by a logistic link,

$$p(Y_{ij} = 1 \mid U_i, V_j) = \sigma(U_i^\top V_j), \tag{4.1}$$

the posterior becomes non-Gaussian and analytically intractable. We therefore approximate it with a Laplace approximation, replacing the true posterior with a local Gaussian centered at the mode (the MAP estimate).

Let the log-posterior be:

$$\ell(U_i) = \log p(U_i) + \sum_{j \in \mathcal{D}_t} \log p(Y_{ij} \mid U_i). \tag{4.2}$$

We expand $\ell(U_i)$ around its maximum a posteriori estimate $\hat{U}_i$:

$$\ell(U_i) \approx \ell(\hat{U}_i) - \tfrac{1}{2}(U_i - \hat{U}_i)^\top G (U_i - \hat{U}_i), \tag{4.3}$$

where $G = -\nabla^2_{U_i} \ell(\hat{U}_i)$ is the negative Hessian of the log-posterior. This yields the Gaussian approximation:

$$p(U_i \mid \mathcal{D}_t) \approx \mathcal{N}(\hat{U}_i, G^{-1}). \tag{4.4}$$

In the logistic model,

$$G = I + \sum_j p_j (1 - p_j) V_j V_j^\top, \tag{4.5}$$

where $p_j = \sigma(\hat{U}_i^\top V_j)$ and I arises from the Gaussian prior $p(U_i) = \mathcal{N}(0, I)$.

At each time step t:

1. **Update posterior approximation** Compute $\hat{U}_i$ by maximizing the log-posterior:

$$\hat{U}_i = \arg\max_{U_i} \Big[\sum_{j \in \mathcal{D}_t} Y_{ij} \log \sigma(U_i^\top V_j) + (1 - Y_{ij}) \log(1 - \sigma(U_i^\top V_j)) - \tfrac{1}{2}|U_i|^2 \Big]. \tag{4.6}$$

2. **Compute Hessian at the mode**

$$G = I + \sum_j p_j (1 - p_j) V_j V_j^\top. \tag{4.7}$$

3. **Sample a posterior draw**

$$\tilde{U}_i^{(t)} \sim \mathcal{N}(\hat{U}_i, G^{-1}). \tag{4.8}$$

4. **Select next item**

$$j^* = \arg\max_j \tilde{U}_i^{(t)\top} V_j. \tag{4.9}$$

This algorithm naturally unifies measurement and decision: the posterior (a measurement construct) drives the sampling rule (a decision construct).

Code: Thompson Sampling with Laplace Approximation

```
import torch
from torch.distributions import MultivariateNormal, Bernoulli
from torch.optim import LBFGS

def fit_logistic_map(Y, V, prior_var=1.0, max_iter=50):
    """Estimate MAP and approximate posterior via Laplace."""
    K = V.shape[1]
    U = torch.zeros(K, requires_grad=True)
    opt = LBFGS([U], lr=1.0, max_iter=max_iter)

    def closure():
        opt.zero_grad()
        logits = V @ U
        nll = - (Y * torch.log(torch.sigmoid(logits)) +
                 (1 - Y) * torch.log(1 - torch.sigmoid(logits))).sum()
        nll = nll + 0.5 * (U @ U) / prior_var  # Gaussian prior
        nll.backward()
        return nll

    opt.step(closure)
    with torch.no_grad():
        p = torch.sigmoid(V @ U)
        W = torch.diag(p * (1 - p))
        H = torch.eye(K) / prior_var + V.T @ W @ V
    cov = torch.inverse(H)
    return U.detach(), cov

# Example usage
torch.manual_seed(0)
V = torch.randn(50, 3)
U_true = torch.tensor([1.0, -0.5, 0.7])
Y = Bernoulli(torch.sigmoid(V @ U_true)).sample()

U_map, cov = fit_logistic_map(Y, V)
U_sample = MultivariateNormal(U_map, cov).sample()
scores = (V @ U_sample).detach()
j_star = scores.argmax().item()
print(f"Next item to recommend: {j_star}")
```

Code: Linear Thompson Sampling Simulation

This simulation runs Thompson Sampling with linear utility over multiple rounds, tracking how the posterior concentrates and regret accumulates.

```
#| autorun: true
# Linear Thompson Sampling simulation
np.random.seed(42)

# Setup: 2D user embedding, 30 items
K = 2  # Embedding dimension
n_items = 30
T = 50  # Number of rounds

# True user embedding (unknown to algorithm)
U_true = np.array([1.0, -0.5])

# Item features
V = np.random.randn(n_items, K)
true_probs = 1 / (1 + np.exp(-V @ U_true))
best_item = np.argmax(V @ U_true)

def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

def linear_ts_step(U_hat, U_cov, V_available):
    """Thompson Sampling: sample from posterior, select best item."""
    # Sample from approximate posterior
    U_sample = np.random.multivariate_normal(U_hat, U_cov)
    # Select item with highest sampled utility
    scores = V_available @ U_sample
    return np.argmax(scores), U_sample

```

4.4 Thompson Sampling under Nonlinear Objective

The linear formulation assumes that a user's latent utility varies linearly with item features. In practice, preference landscapes are often **highly nonlinear**: users may favor intermediate difficulty, exhibit saturation, or respond differently to combinations of attributes. To capture such complex relations, we move from a **parametric vector representation** (U_i) to a **nonparametric function representation** (h_i).

A **Gaussian Process (GP)** defines a distribution over functions:

$$h_i \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)), \tag{4.10}$$

where $m(V)$ is the mean function (often assumed to be zero) and $k(V, V')$ is the **kernel**, encoding similarity between items. Intuitively, a GP asserts that **similar items yield similar utilities**, with the kernel controlling smoothness and generalization.

For any finite collection of inputs $V_{1:T}$, the corresponding function values

$$H_{i,1:M} = [h_i(V_1), \dots, h_i(V_M)] \tag{4.11}$$

follow a multivariate normal distribution:

$$H_{i,1:M} \sim \mathcal{N}(m_{1:M}, K_{1:M,1:M}), \tag{4.12}$$

where $(K)_{jj'} = k(V_j, V_{j'})$.

The probability density function (PDF) of a multivariate normal random variable

$$H \sim \mathcal{N}(m, K) \tag{4.13}$$

is given by

$$p(H) = \frac{1}{(2\pi)^{t/2}|K|^{1/2}} \exp\Big(-\tfrac{1}{2}(H-m)^\top K^{-1}(H-m)\Big). \tag{4.14}$$

Taking the logarithm yields

$$\log p(H) = -\tfrac{1}{2}(H-m)^\top K^{-1}(H-m) - \tfrac{1}{2}\log|K| - \tfrac{t}{2}\log(2\pi). \tag{4.15}$$

In most inference contexts, constant terms (those not depending on H) can be dropped, since they cancel when normalizing the posterior.

This nonparametric formulation is advantageous because:

- It **automatically adapts model complexity** to the data through the kernel.
- It **provides closed-form posterior updates** under Gaussian noise.
- It **generalizes across items** by exploiting smoothness in feature space.

The **RBF (Radial Basis Function) kernel** is particularly common:

$$k(V, V') = \sigma^2 \exp\left(-\frac{\|V - V'\|^2}{2\ell^2}\right), \tag{4.16}$$

where ℓ is the **lengthscale** (controlling smoothness) and σ^2 is the **variance** (controlling amplitude). Small lengthscales produce wiggly functions; large lengthscales produce smooth functions.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

Code: Kernel Comparison and GP Prior Samples

```
#| autorun: true
# Visualize RBF kernels with different lengthscales
def rbf_kernel(X1, X2, lengthscale=1.0, variance=1.0):
    """RBF kernel between two sets of points."""
    diff = X1[:, None] - X2[None, :]
    return variance * np.exp(-0.5 * diff**2 / lengthscale**2)

# Create a grid of points
X = np.linspace(-3, 3, 100)

# Compare kernels with different lengthscales
lengthscales = [0.3, 1.0, 3.0]
fig, axes = plt.subplots(2, 3, figsize=(6, 4))

np.random.seed(42)
for i, ls in enumerate(lengthscales):
    # Top row: kernel covariance from center point
    K_center = rbf_kernel(X, np.array([0.0]), lengthscale=ls).flatten()
    axes[0, i].plot(X, K_center, 'b-', linewidth=1.5)
    axes[0, i].set_title(f'$\\ell = {ls}$')
    axes[0, i].set_ylabel('$k(x, 0)$' if i == 0 else '')
    axes[0, i].set_xlabel('$x$')
    axes[0, i].grid(True, alpha=0.3)

    # Bottom row: sample functions from GP prior
    K = rbf_kernel(X, X, lengthscale=ls) + 1e-6 * np.eye(len(X))
    L = np.linalg.cholesky(K)
    for _ in range(5):
        f_sample = L @ np.random.randn(len(X))
        axes[1, i].plot(X, f_sample, alpha=0.6, linewidth=0.8)
    axes[1, i].set_ylabel('$h(x)$' if i == 0 else '')
    axes[1, i].set_xlabel('$x$')
    axes[1, i].set_ylim(-3, 3)
    axes[1, i].grid(True, alpha=0.3)

axes[0, 0].set_title(f'Kernel ($\\ell = 0.3$, wiggly)')
axes[0, 1].set_title(f'Kernel ($\\ell = 1.0$, medium)')
axes[0, 2].set_title(f'Kernel ($\\ell = 3.0$, smooth)')
plt.tight_layout()
plt.show()
print("Top: Kernel covariance from center. Bottom: Random function samples from
 ↪ GP prior.")
print("Smaller lengthscale → more variation; larger lengthscale → smoother
 ↪ functions.")
```

For **binary observations**,

$$p(Y_{ij} = 1 \mid h_i(V_j)) = \sigma(h_i(V_j)), \tag{4.17}$$

the likelihood is non-Gaussian, and the posterior

$$p(h_i \mid \mathcal{D}_t) \propto p(Y_{1:t} \mid h_i) p(h_i) \tag{4.18}$$

becomes **intractable**. To approximate it, we again use the **Laplace approximation**.

Let $H_i = [h_i(V_1), \dots, h_i(V_t)]$. The log-posterior is

$$\ell(H_i) = \log p(Y_{1:t} \mid H_i) - \tfrac{1}{2} H_i^\top K_t^{-1} H_i - \tfrac{1}{2} \log |K_t|. \tag{4.19}$$

Let

$$\hat{H}_i = \arg\max_h \ell(h) \tag{4.20}$$

be the **mode** (MAP).

The multivariate second-order Taylor expansion of a smooth scalar function $\ell(\cdot)$ around $\hat{H}_i$ is

$$\ell(H_i) \approx \ell(\hat{H}_i) + \nabla \ell(\hat{H}_i)^\top (H_i - \hat{H}_i) + \tfrac{1}{2} (H_i - \hat{H}_i)^\top \nabla^2 \ell(\hat{H}_i)(H_i - \hat{H}_i). \tag{4.21}$$

- Because $\hat{H}_i$ is a maximizer, the gradient vanishes:

$$\nabla \ell(\hat{H}_i) = 0. \tag{4.22}$$

- The Hessian at a maximum is **negative semidefinite**:

$$\nabla^2 \ell(\hat{H}_i) \preceq 0. \tag{4.23}$$

So the expansion simplifies to

$$\ell(H_i) \approx \ell(\hat{H}_i) + \tfrac{1}{2} (H_i - \hat{H}_i)^\top \nabla^2 \ell(\hat{H}_i)(H_i - \hat{H}_i). \tag{4.24}$$

Define the **negative Hessian** (a positive semidefinite matrix)

$$W := -\nabla^2 \ell(\hat{H}_i) \succeq 0. \tag{4.25}$$

Then

$$\ell(H_i) \approx \ell(\hat{H}_i) - \tfrac{1}{2} (H_i - \hat{H}_i)^\top W (H_i - \hat{H}_i). \tag{4.26}$$

Laplace's idea is: if the **log** posterior is approximately quadratic, then the **posterior** is approximately Gaussian. Exponentiate the quadratic approximation:

$$p(H_i \mid \mathcal{D}_t) \propto \exp\left(\ell(H_i)\right) \approx \exp\left(\ell(\hat{H}_i) - \tfrac{1}{2} (H_i - \hat{H}_i)^\top W (H_i - \hat{H}_i)\right). \tag{4.27}$$

Since $\exp(\ell(\hat{H}_i))$ is just a constant in H_i, this is proportional to the kernel of a multivariate normal with mean $\hat{H}_i$ and precision W (i.e., covariance W^{-1}):

$$p(H_i \mid \mathcal{D}_t) \approx \mathcal{N}(\hat{H}_i, W^{-1}). \tag{4.28}$$

The resulting approximate posterior can be **projected to any new input** V_* via standard GP conditioning:

$$p(h(V_*) \mid \mathcal{D}_t) \approx \mathcal{N}(\mu_t(V_*), \sigma_t^2(V_*)), \tag{4.29}$$

where

$$\mu_t(V_*) = k_*^\top K_t^{-1} \hat{H}_i, \quad \sigma_t^2(V_*) = k(V_*, V_*) - k_*^\top (K_t + W^{-1})^{-1} k_*. \tag{4.30}$$

where the k_* is the covariance between the new point and previous points.

Once the posterior is approximated, **Thompson Sampling** proceeds as before: sample one plausible function from the posterior, then act optimally with respect to it.

Algorithmically:

1. **Approximate the posterior** $p(h_i \mid \mathcal{D}_t) \approx \mathcal{N}(\hat{H}_i, \Sigma_f)$ via Laplace.
2. **Sample a function realization**

$$\tilde{h}_i \sim \mathcal{GP}(m_t(\cdot), k_t(\cdot, \cdot)), \tag{4.31}$$

   where m_t and k_t are derived from the approximate posterior.
3. **Select the next item**

$$j^* = \arg\max_j \tilde{h}_i(V_j). \tag{4.32}$$

This random sampling induces exploration: uncertain regions of the function space produce high variance in $\tilde{h}_i(V_j)$, encouraging the agent to test them. As more observations are collected, the posterior variance shrinks and the strategy converges toward greedy exploitation.

Below is a minimal illustration using **GP classification with Laplace approximation** and Thompson sampling. (For practical scalability, packages like `gpytorch` or `scikit-learn` can be used.)

Code: GP Thompson Sampling

```
import torch
import math
from torch.distributions import MultivariateNormal, Bernoulli

def rbf_kernel(X1, X2, lengthscale=1.0, variance=1.0):
    diff = X1[:, None, :] - X2[None, :, :]
    dist2 = (diff ** 2).sum(-1)
    return variance * torch.exp(-0.5 * dist2 / lengthscale**2)

def gp_laplace_binary(V, Y, lengthscale=1.0, variance=1.0, max_iter=10):
    N = len(Y)
    K = rbf_kernel(V, V, lengthscale, variance)
    f = torch.zeros(N)
    for _ in range(max_iter):
        p = torch.sigmoid(f)
        W = torch.diag(p * (1 - p))
        z = f + (Y - p) / (p * (1 - p) + 1e-6)
        B = torch.eye(N) + torch.sqrt(W) @ K @ torch.sqrt(W)
        L = torch.linalg.cholesky(B)
        a = torch.cholesky_solve(torch.sqrt(W) @ z, L)
        f = K @ (torch.sqrt(W) @ a)
    W = torch.diag(p * (1 - p))
    Sigma = K - K @ torch.sqrt(W) @ torch.cholesky_solve(torch.sqrt(W) @ K, L)
    return f.detach(), Sigma.detach()

# Example usage
torch.manual_seed(0)
V = torch.linspace(-3, 3, 40).unsqueeze(1)
true_f = torch.sin(V * 1.5)
Y = Bernoulli(torch.sigmoid(true_f)).sample()

f_hat, Sigma = gp_laplace_binary(V, Y)
f_sample = MultivariateNormal(f_hat.squeeze(), Sigma).sample()
j_star = f_sample.argmax().item()
print(f"Next item index: {j_star}")
```

Code: GP Posterior Visualization

The following code visualizes how the GP posterior updates after observing binary responses. The shaded region shows the 95% credible interval.

```
#| autorun: true
# GP posterior visualization with binary observations
def rbf_kernel_np(X1, X2, lengthscale=1.0, variance=1.0):
    diff = X1[:, None] - X2[None, :]
    return variance * np.exp(-0.5 * diff**2 / lengthscale**2)

def gp_laplace_binary_np(X_obs, Y_obs, X_pred, lengthscale=1.0, variance=1.0,
 ↪  max_iter=10):
    """GP classification with Laplace approximation (numpy version)."""
    N = len(Y_obs)
    K = rbf_kernel_np(X_obs, X_obs, lengthscale, variance) + 1e-6 * np.eye(N)
    K_star = rbf_kernel_np(X_pred, X_obs, lengthscale, variance)
    K_ss = rbf_kernel_np(X_pred, X_pred, lengthscale, variance)

    # Newton iteration for MAP
    f = np.zeros(N)
    for _ in range(max_iter):
        p = 1 / (1 + np.exp(-f))
        W = np.diag(p * (1 - p) + 1e-8)
        grad = Y_obs - p
        H = W + np.linalg.inv(K)
        f = f + np.linalg.solve(H, grad)

    # Posterior mean and variance at prediction points
    p = 1 / (1 + np.exp(-f))
    W = np.diag(p * (1 - p) + 1e-8)
    K_inv = np.linalg.inv(K)
    mu_pred = K_star @ K_inv @ f
    cov_pred = K_ss - K_star @ np.linalg.inv(K + np.linalg.inv(W)) @ K_star.T
    return mu_pred, np.sqrt(np.diag(cov_pred) + 1e-8)

# Generate data
np.random.seed(42)
X_grid = np.linspace(-3, 3, 100)
true_f = np.sin(1.5 * X_grid)  # True latent function

# Observe at sparse locations
X_obs = np.array([-2.5, -1.5, -0.5, 0.5, 1.5, 2.5])
f_obs = np.sin(1.5 * X_obs)
Y_obs = (np.random.rand(len(X_obs)) < 1/(1 + np.exp(-f_obs))).astype(float)

# Fit GP
mu, sigma = gp_laplace_binary_np(X_obs, Y_obs, X_grid, lengthscale=1.0)

# Plot
fig, axes = plt.subplots(1, 2, figsize=(6, 3))

# Left: latent function posterior
axes[0].plot(X_grid, true_f, 'k--', label='True $h(x)$', linewidth=1)
axes[0].plot(X_grid, mu, 'b-', label='Posterior mean', linewidth=1.5)
axes[0].fill_between(X_grid, mu - 2*sigma, mu + 2*sigma, alpha=0.3, color='blue',
 ↪  label='95% CI')
axes[0].scatter(X_obs, f_obs, c=['green' if y else 'red' for y in Y_obs],
```

Code: Thompson Sampling Simulation

This simulation shows Thompson Sampling in action, demonstrating how the algorithm balances exploration and exploitation over time.

```
#| autorun: true
# Thompson Sampling simulation with GP
np.random.seed(123)

# True utility function (unknown to algorithm)
def true_utility(x):
    return np.sin(2 * x) + 0.5 * np.cos(x)

# Available items (discrete set)
n_items = 20
items = np.linspace(-2.5, 2.5, n_items)
true_utils = true_utility(items)
best_item = np.argmax(true_utils)

# Thompson Sampling loop
T = 30  # number of rounds
observations_x = []
observations_y = []
chosen_items = []
regret = []
cumulative_regret = 0

for t in range(T):
    if t < 3:
        # Initial random exploration
        j = np.random.randint(n_items)
    else:
        # Fit GP on observations
        X_obs = np.array(observations_x)
        Y_obs = np.array(observations_y)

        # GP posterior
        K = rbf_kernel_np(X_obs, X_obs, lengthscale=0.8) + 1e-4 *
 ↪  np.eye(len(X_obs))
        K_star = rbf_kernel_np(items, X_obs, lengthscale=0.8)
        K_ss = rbf_kernel_np(items, items, lengthscale=0.8)

        # Laplace approximation (simplified)
        f = np.zeros(len(X_obs))
        for _ in range(5):
            p = 1 / (1 + np.exp(-f))
            W = np.diag(p * (1 - p) + 1e-6)
```

Key Takeaway: Thompson Sampling

Thompson Sampling provides a remarkably simple and effective solution to the exploration-exploitation trade-off: **sample a plausible world from your posterior beliefs, then act optimally within it**. This naturally concentrates exploration where uncertainty is highest, without requiring explicit exploration bonuses. Combined with Laplace approximation (for tractable posteriors) and Gaussian Processes (for flexible reward functions), Thompson Sampling provides a complete framework for sequential decision-making under preference uncertainty—connecting directly to the Bayesian inference tools developed in Chapter 2.

4.5 Dueling Bandit

4.5.1 From Multi-Armed Bandits to Dueling Bandits

The multi-armed bandit (MAB) problem involves a gambler deciding which lever to pull on an MAB machine to maximize the winning rate, despite not knowing which machine is the most rewarding. This scenario highlights the need to balance exploration (trying new machines to discover potential higher rewards) and exploitation (using current knowledge to maximize gains). MAB algorithms address this dilemma by making decisions under uncertainty to achieve the best possible outcomes based on gathered data. At the core of the MAB problem is a set of actions, or 'arms,' denoted by $\mathcal{A} = \{1, 2, \dots, K\}$, where K signifies the total number of arms. For each round t, the agent selects an arm $a_t \in \mathcal{A}$ and receives a reward r_t, sampled from an arm-specific, unknown probability distribution. The expected reward of pulling arm a is represented as $\mu_a = \mathbb{E}[r_t|a]$.

The multi-armed bandit framework can be extended in various ways to model more complex scenarios. In the infinite-armed bandit problem, the set of possible arms $\mathcal{A}$ is either very large or infinite. This introduces significant challenges in exploration, as the agent cannot afford to explore each arm even once. Algorithms for infinite-armed bandits typically assume some regularity or structure of the reward function across arms to make the problem tractable. The contextual bandit problem extends the bandit framework by incorporating observable external states or contexts that influence the reward distributions of arms. The agent's task is to learn policies that map contexts to arms to maximize reward. This model is particularly powerful for personalized recommendations, where the context can include user features or historical interactions. In dueling bandit problems, the agent chooses two arms to pull simultaneously and receives feedback only on which of the two is better, not the actual reward values. This pairwise comparison model is especially useful in scenarios where absolute evaluations are difficult, but relative preferences are easier to determine, such as in ranking systems.

Contextual bandits extend the multi-armed bandits by making decisions conditional on the state of the environment and previous observations. The benefit of such a model is that observing the environment can provide additional information, potentially leading to better rewards and outcomes. In each iteration, the agent is presented with the context of the environment, then decides on an action based on the context and previous observations. Finally, the agent ob-

serves the action's outcome and reward. Throughout this process, the agent aims to maximize the expected reward.

In many real-world contexts, one may not have a real-valued reward (or at least a reliable one) associated with a decision. Instead, we may only have observations indicating which of a set of bandits was optimal in a given scenario. The assumption is that within these observations of preferred choices among a set of options, there is an implicit reward or payoff encapsulated in that decision. Consider the following examples:

1. **Dietary preferences**: When providing food recommendations to humans, it is often not possible to quantify an explicit reward from recommending a specific food item. Instead, we can offer meal options and observe which one the person selects.
2. **Video recommendation**: Websites like YouTube and TikTok recommend specific videos to users. It is typically not feasible to measure the reward a person gains from watching a video. However, we can infer that a user preferred one video over another. From these relative preference observations, we can develop a strategy to recommend videos they are likely to enjoy.
3. **Exoskeleton gait optimization**: Tucker et al. (2020) created a framework that uses human-evaluated preferences for an exoskeleton gait algorithm to develop an optimal strategy for the exoskeleton to assist a human in walking. A human cannot reliably produce a numerical value for how well the exoskeleton helped them walk but can reliably indicate which option performed best according to their preferences.

Generally, we assume access to a set of actions. A noteworthy assumption is that any observations we make are unbiased estimates of the payoff. This means that if we observe a human preferred one option over another (or several others), the preferred option had a higher implicit reward or payoff than the alternatives. In the case of dietary preferences, this may mean that a human liked the preferred option; in the case of video recommendations, a user was more entertained, satisfied, or educated by the video they selected than the other options.

The overarching context is that we do not have direct or reliable access to rewards. We may not have a reward at all (for some decisions, it may be impossible to define a real value to the outcome), or it may be noisy (for example, if we ask a human to rate their satisfaction on a scale of 1 to 10). We use relative comparisons to evaluate the best of multiple options in this case. Our goal is to minimize total regret in the face of noisy comparisons. Humans may not always provide consistent observations (since human decision-making is not guaranteed to be consistent). However, we can still determine an optimal strategy with the observed comparisons. We aim to minimize the frequency of sub-optimal decisions according to human preferences. In practice, many formulations of bandits can allow for infinitely many bandits (for example, in continuous-value and high-dimensional spaces). However, this situation can be intractable when determining an optimal decision strategy. With infinite options, how can we always ensure we have chosen the best? We will constrain our bandits to a discrete space to enable efficient exploration. We will assume that we have k bandits, $b_i, i \in [1, k]$, and our task is to choose the one that will minimize regret.

With the framework outlined, we now define our approach more formally. This method was introduced by (Yue et al. 2012), and proofs for the guarantees and derivations of parameters can be found in their work.

To determine the optimal action, we will compare pairwise to ascertain the probability that an action b_i is preferred over another b_j, where $i \neq j$. Concretely, we assume access to a function ϵ that helps determine this probability; in practice, this can be done with an oracle, such as asking a human which of two options they prefer:

$$P(b_i > b_j) = \varepsilon(b_i, b_j) + \frac{1}{2}. \tag{4.33}$$

With this model, three basic properties govern the values provided by ϵ:

$$\epsilon(b_i, b_j) = -\epsilon(b_j, b_i), \epsilon(b_i, b_i) = 0, \epsilon(b_i, b_j) \in \left(-\frac{1}{2}, \frac{1}{2}\right). \tag{4.34}$$

We assume there is a total ordering of bandits, such that $b_i \succ b_j$ implies $\epsilon(b_i, b_j) > 0$. We impose two constraints to properly model comparisons:

- **Strong Stochastic Transitivity**: We must maintain our total ordering of bandits, and as such, the comparison model also respects this ordering:

$$b_i \succ b_j \succ b_k \Rightarrow \epsilon(b_i, b_k) \geq \max\{\epsilon(b_i, b_j), \epsilon(b_j, b_k)\}. \tag{4.35}$$

- **Stochastic Triangle Inequality**: We also impose a triangle inequality, which captures the condition that the probability of a bandit winning (or losing) a comparison will exhibit diminishing returns as it becomes increasingly superior (or inferior) to the competing bandit:

$$b_i \succ b_j \succ b_k \Rightarrow \epsilon(b_i, b_k) \leq \epsilon(b_i, b_j) + \epsilon(b_j, b_k). \tag{4.36}$$

These assumptions may initially seem limiting; however, common models for comparisons satisfy these constraints. For example, the Bradley-Terry Model follows $P(b_i > b_j) = \frac{\mu_i}{\mu_i + \mu_j}$. The Gaussian model with unit variance also satisfies these constraints: $P(b_i > b_j) = P(X_i - X_j > 0)$, where $X_i - X_j \sim N(\mu_i - \mu_j, 2)$.

Code: Preference Matrix Visualization

```
#| autorun: true
# Generate and visualize a preference matrix from Bradley-Terry model
def bradley_terry_matrix(strengths):
    """Generate preference matrix from Bradley-Terry strengths."""
    n = len(strengths)
    P = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i != j:
                P[i, j] = strengths[i] / (strengths[i] + strengths[j])
            else:
                P[i, j] = 0.5
    return P

# Example: 5 bandits with different strengths
np.random.seed(42)
strengths = np.array([3.0, 2.5, 2.0, 1.5, 1.0])
labels = ['A', 'B', 'C', 'D', 'E']
P = bradley_terry_matrix(strengths)

fig, axes = plt.subplots(1, 2, figsize=(6, 3))

# Preference matrix heatmap
im = axes[0].imshow(P, cmap='RdYlGn', vmin=0, vmax=1)
axes[0].set_xticks(range(len(labels)))
axes[0].set_yticks(range(len(labels)))
axes[0].set_xticklabels(labels)
axes[0].set_yticklabels(labels)
axes[0].set_xlabel('Opponent $j$')
axes[0].set_ylabel('Bandit $i$')
axes[0].set_title('Preference Matrix $P_{ij}$')
for i in range(len(labels)):
    for j in range(len(labels)):
        axes[0].text(j, i, f'{P[i,j]:.2f}', ha='center', va='center', fontsize=7)
plt.colorbar(im, ax=axes[0], shrink=0.8)

# Epsilon matrix (Delta = P - 0.5)
Delta = P - 0.5
im2 = axes[1].imshow(Delta, cmap='RdBu', vmin=-0.5, vmax=0.5)
axes[1].set_xticks(range(len(labels)))
axes[1].set_yticks(range(len(labels)))
axes[1].set_xticklabels(labels)
axes[1].set_yticklabels(labels)
axes[1].set_xlabel('Opponent $j$')
axes[1].set_ylabel('Bandit $i$')
axes[1].set_title('$\\epsilon_{ij} = P_{ij} - 0.5$')
for i in range(len(labels)):
    for j in range(len(labels)):
        axes[1].text(j, i, f'{Delta[i,j]:.2f}', ha='center', va='center',
    ↪ fontsize=7)
plt.colorbar(im2, ax=axes[1], shrink=0.8)

plt.tight_layout()
```

To accurately model the preferences between bandits in our framework of pairwise bandit comparisons and regret, we must track certain parameters in our algorithm. First, we will maintain a running empirical estimate of the probability of bandit preferences based on our observations. It is important to note that we do not have direct access to an ϵ function. Instead, we must present two bandits to a human, who selects a winner. To do this, we define:

$$\hat{P}_{i,j} = \frac{\#b_i \text{ wins}}{\#\text{comparisons between } i \text{and } j}. \tag{4.37}$$

We will also compute confidence intervals at each timestep for each of the entries in $\hat{P}$ as

$$\hat{C}_t = \left(\hat{P}_t - c_t, \hat{P}_t + c_t\right), \tag{4.38}$$

where $c_t = \sqrt{\frac{4\log(\frac{1}{\delta})}{t}}$. Note that $\delta = \frac{1}{TK^2}$, where T is the time horizon and K is the number of bandits.

Previously, we discussed approaches for finding the best action in a specific context. Now, we consider changing contexts, which means there is no longer a static hidden preference matrix P. Instead, at every time step, there is a preference matrix P_C depending on context C. We consider a context C and a preference matrix P_C to be chosen by nature as a result of the given environment (Yue et al., 2012). The goal of a contextual bandits algorithm is to find a policy π that maps contexts to a Von Neumann winner distribution over our bandits. That is, our policy π should map any context to some distribution over our bandits such that sampling from that distribution is preferred to a random action for that context.

4.5.2 Regret

The agent aims to pick a sequence of arms $(a_1, a_2, \ldots, a_T)$ across a succession of time steps $t = 1$ to $t = T$ to maximize the total accumulated reward. Formally, the strategy seeks to maximize the sum of the expected rewards: $\max_{a_1,\ldots,a_T} \mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$. Regret is defined as the difference between the cumulative reward that could have been obtained by always pulling the best arm (in hindsight, after knowing the reward distributions) and the cumulative reward actually obtained by the algorithm. Formally, if μ^* is the expected reward of the best arm and μ_{a_t} is the expected reward of the arm chosen at time t, the regret after T time steps is given by $R(T) = T \cdot \mu^* - \sum_{t=1}^{T} \mu_{a_t}$. The objective of a bandit algorithm is to minimize this regret over time, effectively learning to make decisions that are as close as possible to the decisions of an oracle that knows the reward distributions beforehand. Low regret indicates an algorithm that has often learned to choose well-performing arms, balancing the exploration of unknown arms with the exploitation of arms that are already known to perform well. Thus, an efficient bandit algorithm exhibits sub-linear regret growth, meaning that the average regret per round tends to zero as the number of rounds T goes to infinity: $\lim_{T\to\infty} \frac{R(T)}{T} = 0$. Minimizing regret is a cornerstone in the design of bandit algorithms, and its analysis helps in understanding the long-term efficiency and effectiveness of different bandit strategies.

As previously discussed, our goal is to select the bandit that minimizes a quantity that reflects regret or the cost of not selecting the optimal bandit at all times. We can leverage our comparison model to define a quantity for regret over some time horizon T, which is the number of decisions we make (selecting what we think is the best bandit at each iteration). Assuming we know the best bandit b^* (and we know that there *is* a best bandit, since there is a total ordering of our discrete bandits), we can define two notions of regret:

- Strong regret: aims to capture the fraction of users who would prefer the optimal bandit b^* over the *worse* of the options b_1, b_2 we provide at a given step:$R_T = \sum_{t=1}^{T} \max\left\{\epsilon(b^*, b_1^{(t)}), \epsilon(b^*, b_2^{(t)})\right\}$
- Weak regret: aims to capture the fraction of users who would prefer the optimal bandit b^* over the *better* of the options b_1, b_2 we provide at a given step:$\tilde{R}_T = \sum_{t=1}^{T} \min\left\{\epsilon(b^*, b_1^{(t)}), \epsilon(b^*, b_2^{(t)})\right\}$

4.5.3 Winner Concepts

The best bandit described in our regret definition is called a **Condorcet Winner**. This is the strongest form of winner. It's the action A_i which is preferred to each other action A_j with $p > 0.5$ in a head-to-head election. While the above introduced notions of regret assume an overall best bandit to exist, there might be settings, where no bandit wins more than half head-to-head duels. A set of actions without a Condorcet winner is described by the following preference matrix, where each entry Δ_{jk} is $p(j \succ k) - 0.5$, the probability that action j is preferred over action k minus 0.5. There is no Condorcet winner as there is no action that is preferred with $p > 0.5$ over all other actions. Imagine, you want to find the best pizza to eat (=action). There may not be a pizza that wins more than half of the head-to-head duels against every other pizza.

However, we might still have an intuition of the best pizza. Therefore Sui et al., 2018 introduce the concepts of different *winners* in dueling bandit problems (Sui et al. 2018). In this example, we might define the best pizza as the most popular one. We call the Pizza receiving the most votes in a public vote the **Borda Winner**, or formally, Borda winner $j = \arg\max_{i \in A, i \neq j} (\sum p(j \succ i))$. In contrast to the Condorcet Winner setting, there is always guaranteed to be one or more (in the case of a tie) Borda winners for a set of actions. However - if there is a Condorcet Winner, this might not necessarily be the same as a Borda Winner: In our Pizza example, a Pepperoni Pizza might win more than half of its head-to-head duels, while the Cheese-Pizza is still the most popular in a public poll.

A more generic concept of winner is the **Von Neumann Winner**, which describes a probability distribution rather than a single bandit winner. A Von Neumann winner simply prescribes a probability distribution W such that sampling from this distribution 'beats' an action from the random uniform distribution with $p > 0.5$. In our pizza example, this would correspond to trusting a friend to order whichever Pizza he likes, because this may still be preferred to ordering randomly. Formally, W is a Von Neumann if $(j \sim W, k \sim R)[p(p(j \succ k) > 0.5) > 0.5]$ where R describes the uniform probability distribution over our actions. The concept of a

Von Neumann winner is useful in contextual bandits, which will be introduced later. In these settings, the preference matrix depends on different context, which may have different Borda winners, just as different parties may vote for different pizzas.

No Universal Winner Concept

The three winner concepts—Condorcet, Borda, and von Neumann—can identify **different** items as the "best," and a Condorcet winner may not even exist. The choice of winner concept is therefore a **normative design decision**, not a purely technical one: it determines what "regret" means and what the algorithm optimizes for. This parallels a deeper tension formalized by Arrow's impossibility theorem (): there is no aggregation rule that satisfies all desirable properties simultaneously. When deploying dueling bandit algorithms, practitioners should explicitly choose a winner concept that matches their application's values—majority dominance (Condorcet), average popularity (Borda), or worst-case robustness (von Neumann).

	A	B	C	D	E	F
A	0	**0.03**	**-0.02**	0.06	0.10	0.11
B	-0.03	0	**0.03**	0.05	0.08	0.11
C		-0.03	0	0.04	0.07	0.09
D	-0.06	-0.05	-0.04	0	0.05	0.07
E	-0.10	-0.08	-0.07	-0.05	0	0.03
F	-0.11	-0.11	-0.09	-0.07	-0.03	0

Figure 4.1
Violation of Condorcet Winner. Highlighted entries are different from Table 1. No Condorcet winner exists as no arm could beat every other arm.

Code: Pizza Voting Example - When Winners Diverge

This example demonstrates a scenario where three groups have different pizza preferences, resulting in no Condorcet winner. The Borda winner and Von Neumann winner provide alternative solutions.

```
#| autorun: true
# Pizza voting example: 3 pizzas, 3 voter groups with cyclic preferences
# Group 1 (40%): Pepperoni > Cheese > Veggie
# Group 2 (35%): Cheese > Veggie > Pepperoni
# Group 3 (25%): Veggie > Pepperoni > Cheese

pizzas = ['Pepperoni', 'Cheese', 'Veggie']
n = len(pizzas)

# Construct preference matrix from group preferences
# P[i,j] = probability that pizza i beats pizza j
group_prefs = [
    [0, 1, 2],  # Group 1: Pepperoni(0) > Cheese(1) > Veggie(2)
    [1, 2, 0],  # Group 2: Cheese > Veggie > Pepperoni
    [2, 0, 1],  # Group 3: Veggie > Pepperoni > Cheese
]
group_weights = [0.40, 0.35, 0.25]

P = np.zeros((n, n))
for g, (pref, w) in enumerate(zip(group_prefs, group_weights)):
    for rank_i, i in enumerate(pref):
        for rank_j, j in enumerate(pref):
            if rank_i < rank_j:  # i is ranked higher than j
                P[i, j] += w

# Fill diagonal with 0.5
for i in range(n):
    P[i, i] = 0.5

print("Preference Matrix P[i,j] = P(pizza i beats pizza j):")
print(f"           {pizzas[0]:>10} {pizzas[1]:>10} {pizzas[2]:>10}")
for i, name in enumerate(pizzas):
    print(f"{name:>10}", end="")
    for j in range(n):
        print(f" {P[i,j]:>10.2f}", end="")
    print()

# Check for Condorcet winner
print("\n--- Winner Analysis ---")
condorcet = None
for i in range(n):
    if all(P[i, j] > 0.5 for j in range(n) if j != i):
        condorcet = pizzas[i]
        break
print(f"Condorcet winner: {condorcet if condorcet else 'NONE (no pizza beats all
 ↪  others!)'}")

# Borda scores (sum of win probabilities)
borda_scores = [sum(P[i, j] for j in range(n) if j != i) for i in range(n)]
borda_winner = pizzas[np.argmax(borda_scores)]
print(f"Borda scores: {dict(zip(pizzas, [f'{s:.2f}' for s in borda_scores]))}")
print(f"Borda winner: {borda_winner}")

```

Code: Computing Different Winner Types

```
#| autorun: true
def condorcet_winner(P):
    """Return Condorcet winner index, or None if doesn't exist."""
    n = P.shape[0]
    for i in range(n):
        if all(P[i, j] > 0.5 for j in range(n) if j != i):
            return i
    return None

def borda_winner(P):
    """Return Borda winner index (highest sum of pairwise win probs)."""
    n = P.shape[0]
    scores = [sum(P[i, j] for j in range(n) if j != i) for i in range(n)]
    return np.argmax(scores)

def von_neumann_winner(P, max_iter=1000):
    """Compute Von Neumann (minimax) winner distribution via fictitious play."""
    n = P.shape[0]
    # Payoff matrix: A[i,j] = P[i,j] - 0.5 (zero-sum game)
    A = P - 0.5
    # Fictitious play to find equilibrium
    counts = np.ones(n)
    for _ in range(max_iter):
        # Best response to current empirical distribution
        dist = counts / counts.sum()
        payoffs = A @ dist
        best = np.argmax(payoffs)
        counts[best] += 1
    return counts / counts.sum()

# Test on various preference matrices
print("Example 1: Clear Condorcet winner")
P1 = np.array([[0.5, 0.7, 0.8],
               [0.3, 0.5, 0.6],
               [0.2, 0.4, 0.5]])
c1 = condorcet_winner(P1)
b1 = borda_winner(P1)
v1 = von_neumann_winner(P1)
print(f"  Condorcet: Arm {c1}, Borda: Arm {b1}, VN dist: {v1.round(2)}")

print("\nExample 2: No Condorcet winner (cyclic)")
P2 = np.array([[0.5, 0.6, 0.4],
               [0.4, 0.5, 0.7],
               [0.6, 0.3, 0.5]])
c2 = condorcet_winner(P2)
b2 = borda_winner(P2)
v2 = von_neumann_winner(P2)
print(f"  Condorcet: {'None' if c2 is None else f'Arm {c2}'}, Borda: Arm {b2}, VN
 ↪  dist: {v2.round(2)}")

print("\nExample 3: Condorcet and Borda differ")
P3 = np.array([[0.5, 0.51, 0.51, 0.51],
               [0.49, 0.5, 0.9, 0.9],
```

Next, we introduce two performance measures for the planner. The **asymptotic ex-post regret** is defined as

$$\text{Regret}(\mu_1, \ldots \mu_K) = T \cdot \max_i \mu_i - \sum_{i=1}^{T} E[\mu_{I_t}]. \tag{4.39}$$

Intuitively, this represents the difference between the reward achieved by always taking the action with the highest possible reward and the expected welfare of the recommendation algorithm (based on the actions it recommends at each timestep).

We also define a weaker performance measure, the **Bayesian regret**, which is defined as

$$\text{Bayesian regret} = E_{\mu_1,\ldots,\mu_K \sim \text{Prior}} \left[\text{Regret}\left(\mu_1, \ldots, \mu_K\right)\right] \tag{4.40}$$

With a Bayesian optimal policy, we would like either definition of regret to vanish as $T \to \infty$; we are considering "large-market optimal" settings where there are many short-lived, rather than a few long-term, users. Note the fact that ex-post regret is prior-free makes it robust to inaccuracies on the prior.

4.5.4 Acquisition Functions

Various strategies have been developed to balance the exploration-exploitation trade-off. These strategies differ in selecting arms based on past experiences and rewards.

4.5.4.1 Classical Acquisition Functions

Uniform acquisition function is the most straightforward approach where each arm is selected uniformly randomly over time. This strategy does not consider the past rewards and treats each arm equally promising regardless of the observed outcomes. It is a purely explorative strategy that ensures each arm is sampled enough to estimate its expected reward, but it does not exploit the information to optimize rewards. In mathematical terms, if $N_t(a)$ denotes the number of times arm a has been selected up to time t, the Uniform Strategy would ensure that $N_t(a) \approx \frac{t}{K}$ for all arms a as t grows large: $P(a_t = a) = \frac{1}{K}$

The **Epsilon Greedy** is a popular method that introduces a balance between exploration and exploitation. With a small probability ϵ, it explores by choosing an arm at random, and with a probability $1 - \epsilon$, it exploits by selecting the arm with the highest estimated reward so far. This strategy incrementally favors actions that have historically yielded higher rewards, but still allows for occasional exploration to discover better options potentially. The parameter ϵ is chosen based on the desired exploration level, often set between 0.01 and 0.1.

$$P(a_t = a) = \begin{cases} \frac{\epsilon}{K} + 1 - \epsilon & \text{if } a = \arg\max_{a'} \hat{\mu}_{a'} \\ \frac{\epsilon}{K} & \text{otherwise} \end{cases} \tag{4.41}$$

Upper Confidence Bound (UCB) acquisition function takes a more sophisticated approach to the exploration-exploitation dilemma. It selects arms based on both the estimated rewards and the uncertainty or variance associated with those estimates. Specifically, it favors arms with high upper confidence bounds on the estimated rewards, which is a sum of the estimated mean and a confidence interval that decreases with the number of times the arm has been played. This ensures that arms with less certainty (those played less often) are considered more often, naturally balancing exploration with exploitation as the uncertainty is reduced over time.

$$P(a_t = a) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} \left(\hat{\mu}_{a'} + \sqrt{\frac{2 \ln t}{N_t(a')}} \right) \\ 0 & \text{otherwise} \end{cases} \tag{4.42}$$

4.5.4.2 Interleaved Filter

This algorithm tries to find the best bandit (Condorcet Winner) in a discrete, limited bandit-space via pairwise comparisons of the bandits. We will now introduce the algorithm for the Interleaved Filter as provided in (Yue et al. 2012) to solve a dueling bandit setup. It starts with a randomly defined *best bandit* $\hat{b}$ and iteratively compares it to set W containing the remaining bandits b resulting in winning probabilities $\hat{P}_{\hat{b},b}$ and confidence interval $\hat{C}_{\hat{b},b}$. If a bandit b is *confidently worse* than $\hat{b}$, it is removed from W. If a bandit b' is *confidently better* than $\hat{b}$, it is set as new *best bandit* $\hat{b}$ and bandit $\hat{b}$ as well as every other bandit b *worse* than $\hat{b}$ are removed from W. This is done, until W is empty, leaving the final $\hat{b}$ as the predicted best bandit.

input: T, $B = \{b_1, \dots, b_k\}$ $\delta \leftarrow 1/(TK^2)$ Choose $\hat{b} \in B$ randomly $W \leftarrow \{b_1, \dots, b_k\} \backslash \{\hat{b}\}$ $\forall b \in W$, maintain estimate $\hat{P}_{\hat{b},b}$ of $P(\hat{b} > b)$ according to (6) $\forall b \in W$, maintain $1 - \delta$ confidence interval $\hat{C}_{\hat{b},b}$ of $\hat{P}_{\hat{b},b}$ according to (7), (8) compare $\hat{b}$ and b update $\hat{P}_{\hat{b},b}$, $\hat{C}_{\hat{b},b}$ $W \leftarrow W \backslash \{b\}$

$W \leftarrow W \backslash \{b\}$ $\hat{b} \leftarrow b'$, $W \leftarrow W \backslash \{b'\}$ $\forall b \in W$, reset $\hat{P}_{\hat{b},b}$ and $\hat{C}_{\hat{b},b}$ $\hat{T} \leftarrow$ Total Comparisons Made $(\hat{b}, \hat{T})$

Parameter Initialization In lines 1-6 of the algorithm, we take the inputs and first compute the value δ which is used to compute our confidence intervals. We select an initial guess of an optimal bandit $\hat{b}$ by uniformly sampling from all bandits $\mathcal{B}$. We also keep a running set of bandit candidates W, which is initialized to be $\mathcal{B} \setminus \{\hat{b}\}$. At this point, we also initialize our empirical estimates for $\hat{P}, \hat{C}$.

Next, we will repeat several steps until our working set of bandit candidates W is empty.

Update Estimates Based on Comparisons The first step at each iteration (lines 8-11) is to look at all candidates in W, and compare them to our current guess $\hat{b}$ using an oracle (e.g. by asking a human which of $\hat{b}$ or $b \in W$ is preferred). With this new set of wins and comparisons, we update our estimates of $\hat{P}, \hat{C}$.

Prune Suboptimal Bandits In lines 12–13, with updated comparison win probabilities and corresponding confidence intervals, we can remove bandit candidates from W that we are *confident* $\hat{b}$ is better than. The intuition here is that we are mostly sure that our current best guess is better than some of the candidates, and we don't need to consider those candidates in future iterations.

Check for Better Bandits from Candidate Set Now that our candidate set of bandits may be smaller, in lines 15–21 we check if there are any bandits b' that we are *confident* are better than our current best guess. If we do find such a candidate, we remove bandits which $\hat{P}$ indicates b is *likely* worse than $\hat{b}$. Note that in this step, we do not require the probability to be outside the confidence interval, since we already found one we believe to be significantly closer to optimal than our current best guess.

Once we remove the candidates *likely* worse than $\hat{b}$, we crown b' as the new best guess, e.g. $\hat{b} := b'$. Consequently, we remove b' from W and reset our empirical win counters $\hat{P}, \hat{C}$.

With this algorithm defined, let us look at some provisions of the method with respect to identifying the optimal strategy. Note that the proofs and derivations for these quantities are provided in (Yue et al. 2012).

First, the method guarantees that for the provided time horizon T, the algorithm returns the correct bandit with probability $P \geq 1 - \frac{1}{T}$. It is interesting and useful to note that if one has a strict requirement for the probability of identifying the correct bandit, one can compute the time horizon T that guarantees this outcome at that probability. Furthermore, a time horizon of 1 leaves no probabilistic guarantee of a successful outcome, and increasing T has diminishing returns. Second, in the event that the algorithm returns an incorrect bandit, the maximal regret incurred is linear with respect to T, e.g. $(O)(T)$. This is also a useful provision as it allows us to estimate the overall cost in the worst case outcome. Based on these two provisions, we can compute the expected cumulative regret from running the Interleaved Filter algorithm, which is:

$$\mathbb{E}\left[R_T\right] \leq \left(1 - \frac{1}{T}\right) \mathbb{E}\left[R_T^{IF}\right] + \frac{1}{T}\mathcal{O}(T) = \mathcal{O}\left(\mathbb{E}\left[R_T^{IF}\right] + 1\right) \tag{4.43}$$

Interestingly, the original work shows that these bounds hold for both strong and weak regret. As demonstrated, the Interleaved Filter algorithm [fig-if] provides a robust method to ascertain the optimal bandit or strategy given a set of options and only noisy comparisons. In most real-world scenarios for modeling human preferences, it is not possible to observe a real-world reward value, or at least a reliable one and as such this method is a useful way to properly model human preferences.

Furthermore, the algorithm provides strong guarantees for the probability of selecting the correct bandit, maximal regret, and the number of comparisons required. It is even more impressive that the method can do so without severely limiting constraints; as demonstrated, the most commonly used models satisfy the imposed constraints.

As we look to model human preferences, we can certainly leverage this method for k-armed dueling bandits to identify the best strategy to solve human-centric challenges, from video recommendation to meal selection and exoskeleton-assisted walking.

Code: Interleaved Filter Implementation

```
#| autorun: true
class InterleavedFilter:
    """Interleaved Filter algorithm for finding Condorcet winner."""

    def __init__(self, n_arms, delta=0.1):
        self.n_arms = n_arms
        self.delta = delta
        self.reset()

    def reset(self):
        self.best = np.random.randint(self.n_arms)
        self.candidates = set(range(self.n_arms)) - {self.best}
        self.wins = {j: 0 for j in self.candidates}
        self.counts = {j: 0 for j in self.candidates}
        self.comparisons = 0

    def get_confidence(self, t):
        if t == 0:
            return 1.0
        return np.sqrt(4 * np.log(1 / self.delta) / t)

    def step(self, oracle_fn):
        """Perform one step: compare best against a candidate."""
        if not self.candidates:
            return True  # Done

        # Pick a candidate to compare
        candidate = list(self.candidates)[self.comparisons %
    ↪ len(self.candidates)]

        # Query oracle (returns 1 if best wins, 0 if candidate wins)
        result = oracle_fn(self.best, candidate)
        self.comparisons += 1
        self.counts[candidate] += 1
        if result == 1:
            self.wins[candidate] += 1

        # Update estimates and prune
        to_remove = []
        for j in list(self.candidates):
            if self.counts[j] > 0:
                p_hat = self.wins[j] / self.counts[j]
                c = self.get_confidence(self.counts[j])

                # If best confidently beats j, remove j
                if p_hat - c > 0.5:
                    to_remove.append(j)
                # If j confidently beats best, j becomes new best
                elif p_hat + c < 0.5:
                    self.candidates.discard(j)
                    self.candidates.add(self.best)
                    self.best = j
                    # Reset counts for new comparisons
```

Code: Comparing Dueling Bandit Algorithms

```
#| autorun: true
# Compare Interleaved Filter vs Random exploration
np.random.seed(123)

def run_experiment(n_arms, true_best, n_trials=20, max_comparisons=200):
    """Run multiple trials and measure sample complexity."""
    # Interleaved Filter
    if_results = []
    for _ in range(n_trials):
        algo = InterleavedFilter(n_arms, delta=0.1)
        for step in range(max_comparisons):
            done = algo.step(oracle)
            if done:
                break
        if_results.append((algo.best == true_best, algo.comparisons))

    # Random exploration baseline
    rand_results = []
    for _ in range(n_trials):
        pair_counts = np.zeros((n_arms, n_arms))
        pair_wins = np.zeros((n_arms, n_arms))
        for comp in range(max_comparisons):
            i, j = np.random.choice(n_arms, 2, replace=False)
            result = oracle(i, j)
            pair_counts[i, j] += 1
            pair_counts[j, i] += 1
            if result == 1:
                pair_wins[i, j] += 1
            else:
                pair_wins[j, i] += 1

        # Estimate Condorcet winner
        P_hat = np.where(pair_counts > 0, pair_wins / pair_counts, 0.5)
        best_guess = None
        for i in range(n_arms):
            if all(P_hat[i, j] > 0.5 for j in range(n_arms) if j != i):
                best_guess = i
                break
        if best_guess is None:
            best_guess = np.argmax([sum(P_hat[i, :]) for i in range(n_arms)])
        rand_results.append((best_guess == true_best, max_comparisons))

    return if_results, rand_results

if_res, rand_res = run_experiment(5, 0)

# Summarize
if_acc = np.mean([r[0] for r in if_res])
if_comps = np.mean([r[1] for r in if_res])
rand_acc = np.mean([r[0] for r in rand_res])
rand_comps = np.mean([r[1] for r in rand_res])

print("Algorithm Comparison (20 trials, 5 arms):")
```

4.5.4.3 *Dueling Bandit Gradient Descent*

The Interleaved Filter works well when the action space is finite, but many practical settings involve **continuous** parameter spaces. For example, a search engine may be parameterized by a weight vector $w \in \mathbb{R}^d$ controlling how different ranking signals are combined, and the goal is to find the w that produces the most preferred results. Dueling Bandit Gradient Descent (DBGD) (Yue and Joachims 2009) extends the dueling bandit framework to this continuous setting by performing *gradient-free optimization* using only preference feedback.

The idea is intuitive: at each step, compare the current parameters w_t against a random perturbation $w'_t = w_t + \delta u_t$. If users prefer the perturbed system, take a step in that direction. This is analogous to stochastic gradient descent, but using preference comparisons instead of gradient evaluations.

input: γ, δ, w_1

Sample unit vector u_t uniformly

$w'_t \leftarrow P_W(w_t + \delta u_t)$

Compare w_t and w'_t

$w_{t+1} \leftarrow P_W(w_t + \gamma u_t)$

$w_{t+1} \leftarrow w_t$

We first choose exploration step length δ, exploitation step length γ, and starting point (in unit-sphere) w_1. Choose a query and sample a random unit vector u_t. We duel w_t and w'_t, where w_t is our current point in the sphere, and w'_t is our exploratory comparison, which is generated by taking a random step of length δ, such that $w'_t = w_t + \delta u_t$. The objective of this duel is to ascertain the binary preference of users with respect to the results yielded by the IR systems parameterized by w_t and w'_t respectively, taking query q_t as an input. The parameters that get the majority of the votes in the head to head win. If w_t wins, then we keep the parameters for the next iteration. If w'_t wins the duel, we update our parameters in the direction of u_t by taking a step of length γ. Note that the algorithm describes projection operation $P_W(\vec{v})$. Since u_t is chosen randomly, $w_t + \delta u_t$ or $w_t + \gamma u_t$ could exist outside of the unit sphere where all possible parameter configurations lie. In this case, we simply project the point back onto the sphere using said projection $P_W(\vec{v})$.

Yue and Joachims show that this algorithm has sublinear regret in T, the number of iterations. We note that the algorithm assumes that there exists a hidden reward function $R(w)$ that maps system parameters w_t to a reward value which is smooth and strictly concave over the input space W.

Lastly, we would also like to give motivation behind δ and γ being different values. We need a δ that is sufficiently large that the comparison between a system parameterized by w_t and w'_t is meaningful. On the other hand, we may wish to take a smaller step in the direction of w'_t during our update step, as during a duel, we only score w_t against w'_t over the results on one

query q_t. Having $\delta > \gamma$ allows us to get reward signal from meaningfully different points while also updating our belief of the best point w_{best} gradually.

Sparring EXP4

While DBGD handles continuous action spaces, many real-world dueling problems also involve **context**—the best action depends on the situation. For example, the best search ranking for a query about "python" differs depending on whether the user is a programmer or a pet owner. Sparring EXP4 (Zoghi et al. 2015) addresses this *contextual dueling bandit* setting by adapting the EXP4 algorithm (originally designed for contextual bandits with scalar rewards) to preference feedback.

The key idea is to run **two instances** of the EXP4 algorithm against each other. Each instance maintains embeddings for the actions and produces a probability distribution over choices given the current context. When a context arrives, each instance samples an action, the two actions are dueled, and the winner receives a positive reward while the loser receives a negative reward proportional to the preference strength. This "sparring" approach elegantly converts pairwise preference feedback into the scalar reward signals that EXP4 requires.

4.5.4.4 Feel-good Thompson Sampling

Feel-good Thompson Sampling (Zhang 2021) offers an alternative approach to the contextual dueling bandit setting, connecting back to the Thompson Sampling framework developed earlier in this chapter. Like Sparring EXP4, it handles context-dependent preferences, but it uses posterior sampling rather than adversarial learning. The algorithm aims to find a Von Neumann winner by minimizing cumulative average regret:

$$\text{Regret}(T) := \sum_{t=1}^{T} \left[r_*(x_t, a_t^*) - \frac{r_*(x_t, a_t^1) + r_*(x_t, a_t^2)}{2} \right], \tag{4.44}$$

where $r_*(x_t, a_t)$ is the true, hidden reward function of a context x_t and action a_t. Thompson sampling is an iterative process of receiving preference over two actions, each maximizing a different approximation of the reward function based on past data and adding this new information to the data.

Finding good approximations of the reward function at time t is done by sampling two reward function parameters $\theta_t^{j=1}$ and $\theta_t^{j=2}$ from a posterior distribution based on all previous data $p_j(\cdot \mid S_{t-1})$. This posterior distribution is proportional to the multiplication of the prior and the likelihood function, which is a Gaussian in standard Thompson sampling. In Feel-Good Thompson sampling, an additional term called "Feel-good exploration" encourages parameters θ with a large maximum reward in previous rounds. This change to the likelihood function may increase probabilities in uncertain areas, thus exploring those regions. All that's left is to

select an action maximizing each reward function approximation and receive a preference y_t on one of them to add the new information to the dataset(Zhang 2021).

Initialize $S_0 = \emptyset$. Receive prompt x_t and action space $\mathcal{A}_t$. Sample model parameter θ_t^j from the posterior distribution $p^j(\cdot \mid S_{t-1})$ Select response $a_t^j = \arg\max_{a \in \mathcal{A}_t} \langle \theta_t^j, \phi(x_t, a) \rangle$. Receive preference y_t. Update dataset $S_t \leftarrow S_{t-1} \cup \{(x_t, a_t^1, a_t^2, y_t)\}$.

Key Takeaway: Choosing an Algorithm

The dueling bandit algorithms covered above address different problem settings:

Algorithm	Action Space	Context?	Winner Concept	Key Strength
Interleaved Filter	Finite	No	Condorcet	Simple, provable
DBGD	Continuous	No	Reward maximizer	Gradient-free optimization
Sparring EXP4	Finite	Yes	Copeland	Adversarial robustness
Feel-good TS	Finite	Yes	Von Neumann	Bayesian, posterior sampling

When choosing an algorithm, consider: (1) Is the action space discrete or continuous? (2) Does the optimal action depend on context? (3) What computational resources are available? Thompson Sampling-based methods are generally preferred when a good posterior approximation is available, while adversarial methods like Sparring EXP4 are more robust when the preference model is misspecified.

4.5.5 Applications

The dueling bandit and Thompson Sampling frameworks developed above have broad applicability whenever systems must learn from preference feedback rather than scalar rewards. We highlight two case studies that illustrate how these methods connect to the book's central themes.

4.5.5.1 Case Study 1: LLM Response Selection as a Dueling Bandit

Consider an LLM serving system that generates K candidate responses to each user prompt and must select which to display. This is exactly a contextual dueling bandit problem:

- **Context**: The user prompt x_t and its embedding
- **Actions**: The K candidate responses $\{y_1, \ldots, y_K\}$
- **Preference feedback**: The user indicates which response they prefer (via explicit rating, continued engagement, or regeneration requests)
- **Objective**: Maximize user satisfaction while learning which response characteristics users value

The preference model is Bradley-Terry over response embeddings: $p(y_j \succ y_k \mid x) = \sigma(\beta \log \frac{\pi_\theta(y_j|x)}{\pi_{\text{ref}}(y_j|x)} - \beta \log \frac{\pi_\theta(y_k|x)}{\pi_{\text{ref}}(y_k|x)})$, exactly the DPO formulation from Chapter 2. Thompson Sampling provides a principled way to select which pair of responses to show: sample a reward function from the posterior and present the pair with the highest expected information gain.

This connects the bandit framework to the RLHF pipeline: **online DPO is essentially Thompson Sampling on the Bradley-Terry preference model**. The exploration-exploitation tradeoff determines whether the system shows responses it is confident about (exploitation) or tries novel response styles to learn user preferences (exploration).

```
#| autorun: true
# Simulating LLM response selection as a dueling bandit
rng = np.random.default_rng(42)

# Simulate K=8 response "styles" with true quality scores
K = 8
true_quality = rng.standard_normal(K)
true_quality = np.sort(true_quality)[::-1]  # Best response first

# Bradley-Terry preference probabilities
def bt_prob(q_i, q_j):
    return 1 / (1 + np.exp(-(q_i - q_j)))

# Thompson Sampling with Beta posteriors (simplified)
def run_ts_dueling(n_rounds, true_quality, rng):
    K = len(true_quality)
    wins = np.ones(K)  # Beta prior: alpha=1
    losses = np.ones(K)  # Beta prior: beta=1
    cumulative_regret = []
    total_regret = 0

    for t in range(n_rounds):
        # Sample from posterior (Beta distribution as proxy for quality)
        samples = rng.beta(wins, losses)
        # Select top-2 arms by sampled quality
        top2 = np.argsort(samples)[-2:]
        i, j = top2[0], top2[1]
        # Observe preference (Bradley-Terry)
        p_ij = bt_prob(true_quality[i], true_quality[j])
        if rng.random() < p_ij:
            wins[i] += 1; losses[j] += 1
        else:
            wins[j] += 1; losses[i] += 1
        # Regret: not showing the best response
        best = np.argmax(true_quality)
        regret = true_quality[best] - max(true_quality[i], true_quality[j])
        total_regret += regret
```

```
        cumulative_regret.append(total_regret)
    return cumulative_regret, wins, losses

# Epsilon-greedy baseline
def run_eps_greedy(n_rounds, true_quality, rng, eps=0.1):
    K = len(true_quality)
    wins = np.ones(K); losses = np.ones(K)
    cumulative_regret = []; total_regret = 0

    for t in range(n_rounds):
        if rng.random() < eps:
            pair = rng.choice(K, 2, replace=False)
        else:
            est_quality = wins / (wins + losses)
            pair = np.argsort(est_quality)[-2:]
        i, j = pair[0], pair[1]
        p_ij = bt_prob(true_quality[i], true_quality[j])
        if rng.random() < p_ij:
            wins[i] += 1; losses[j] += 1
        else:
            wins[j] += 1; losses[i] += 1
        best = np.argmax(true_quality)
        regret = true_quality[best] - max(true_quality[i], true_quality[j])
        total_regret += regret
        cumulative_regret.append(total_regret)
    return cumulative_regret, wins, losses

n_rounds = 500
regret_ts, wins_ts, losses_ts = run_ts_dueling(n_rounds, true_quality,
 ↪  np.random.default_rng(0))
regret_eg, wins_eg, losses_eg = run_eps_greedy(n_rounds, true_quality,
 ↪  np.random.default_rng(0))

fig, axes = plt.subplots(1, 2, figsize=(6, 3))
axes[0].plot(regret_ts, label='Thompson Sampling', lw=2)
axes[0].plot(regret_eg, label=' -Greedy ( =0.1)', lw=2, ls='--')
axes[0].set_xlabel('Round'); axes[0].set_ylabel('Cumulative Regret')
axes[0].set_title('LLM Response Selection'); axes[0].legend(); axes[0].grid(True,
 ↪  alpha=0.3)

# Show learned quality estimates
est_ts = wins_ts / (wins_ts + losses_ts)
est_eg = wins_eg / (wins_eg + losses_eg)
x = np.arange(K)
axes[1].bar(x - 0.15, est_ts, 0.3, label='TS estimates', alpha=0.7)
axes[1].bar(x + 0.15, est_eg, 0.3, label=' -Greedy estimates', alpha=0.7)
norm_quality = (true_quality - true_quality.min()) / (true_quality.max() -
 ↪  true_quality.min())
```

```
axes[1].scatter(x, norm_quality, color='red', zorder=5, label='True (normalized)',
 ↪  s=50)
axes[1].set_xlabel('Response Style'); axes[1].set_ylabel('Win Rate / Quality')
axes[1].set_title('Learned Preferences'); axes[1].legend(fontsize=8);
 ↪  axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
print(f"Final regret - TS: {regret_ts[-1]:.1f},  -Greedy: {regret_eg[-1]:.1f}")
```

4.5.5.2 Case Study 2: Clinical Trial Dosing

In personalized medicine, the dueling bandit framework addresses a critical challenge: how to assign drug dosages when the optimal dose depends on individual patient characteristics and can only be evaluated through patient preferences or outcomes.

Consider Warfarin dosing (Bastani and Bayati 2020), where the standard practice is to start with a fixed dose and adjust based on symptoms. This is suboptimal because patients vary significantly in their metabolism. The contextual bandit formulation models this as:

- **Context**: Patient features x_t (age, weight, genetic markers, medication history)
- **Actions**: Dosage levels $a_t \in \{\text{low, medium, high}\}$
- **Reward**: Whether the patient's INR (blood clotting measure) falls in the therapeutic range

Using the linear Thompson Sampling framework from earlier in this chapter, we model the reward as $r(x_t, a_t) = x_t^\top \theta_{a_t}$, where θ_{a_t} captures the relationship between patient features and outcomes for each dosage level. The key insight is that while each patient's outcome reveals information about only one dosage level (we cannot simultaneously try all doses), the *contextual* structure allows information to transfer across similar patients.

This application highlights a tension central to this book: the system must balance **exploration** (trying dosages it is uncertain about to learn their effects) against **exploitation** (prescribing the dosage believed to be best for the current patient). In clinical settings, the cost of exploration is high—a suboptimal dose can cause serious harm—making efficient algorithms like Thompson Sampling especially valuable, since they concentrate exploration where uncertainty is highest rather than exploring uniformly.

4.5.5.3 Broader Applications

Beyond these case studies, bandit methods with preference feedback have been applied to recommendation systems (Zhou et al. 2017; Bouneffouf, Bouzeghoub, and Gançarski 2012), where the cold-start problem for new users is naturally handled by Thompson Sampling's

uncertainty-driven exploration; to dialogue systems (Liu et al. 2018), where response selection is modeled as a contextual bandit; and to search engine optimization (Yue and Joachims 2009), where DBGD tunes ranking parameters from user click preferences. For a comprehensive survey, see Bouneffouf, Rish, and Aggarwal (2020).

4.6 Preferential Bayesian Optimization

The traditional Bayesian optimization (BO) problem is described as follows. There is a black-box objective function $g : \mathcal{X} \to \mathfrak{R}$ defined on a bounded subset $\mathcal{X} \subseteq \mathfrak{R}^q$ such that direct queries to the function are expensive or not possible. However, we would like to solve the global optimization problem of finding $\mathbf{x}_{\min} = \arg\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x})$. This is highly analogous to modeling human preferences, since it is the case that direct access to a human's latent preference function is not possible but we would still like to find its optimum, such as in A/B tests or recommender systems.

We approach this problem for human preferences with *Preferential Bayesian Optimization* (PBO), as the key difference is that we are able to query the preference function through pairwise comparisons of data points, i.e. *duels*. This is a form of indirect observation of the objective function, which models real-world scenarios closely: we commonly need to to optimize a function via data about preferences. With humans, it has been demonstrated that we are better at evaluating differences rather than absolute magnitudes (Kahneman and Tversky 1979) and therefore PBO models can be applied in various contexts.

> **! When Is Pairwise Feedback the Right Choice?**
>
> PBO assumes the oracle can only provide **pairwise comparisons**, not scalar ratings. This is well-justified when absolute evaluations are cognitively difficult or poorly calibrated (e.g., "How good is this robot trajectory on a scale of 1-10?"), but in some domains scalar feedback is readily available and strictly more informative—each scalar observation constrains the reward function more than a binary comparison. When both modalities are feasible, standard Bayesian optimization with scalar feedback typically requires fewer queries to find the optimum. PBO is most valuable when: (1) comparisons are genuinely easier for humans than absolute ratings, (2) the rating scale is arbitrary or poorly anchored across evaluators, or (3) the goal is to find the *best* option rather than estimate the full reward function.

4.6.1 Problem statement

The problem of finding the optimum of a latent preference function defined on $\mathcal{X}$ can be reduced to determining a sequence of duels on $\mathcal{X} \times \mathcal{X}$. From each duel $[\mathbf{x}, \mathbf{x}'] \in \mathcal{X} \times \mathcal{X}$ we obtain binary feedback $\{0, 1\}$ indicating whether or not $\mathbf{x}$ is preferred over $\mathbf{x}'$ ($g(\mathbf{x}) < g(\mathbf{x}')$). We consider that $\mathbf{x}$ is the winner of the duel if the output is $\{1\}$ and that $\mathbf{x}'$ wins the duel if the output is $\{0\}$. The aim is to find $\mathbf{x}_{\min}$ by reducing as much as possible the number of queried duels.

The key idea in PBO is to learn a preference function in the space of duels using a Gaussian process. We define a joint reward $f([\mathbf{x}, \mathbf{x}'])$ on each duel which is never directly observed.

Instead, the feedback we obtain after each pair is a binary output $y \in \{0, 1\}$ indicating which of the two inputs is preferred. One definition of f we will use (though others are possible) is $f([\mathbf{x}, \mathbf{x}']) = g(\mathbf{x}') - g(\mathbf{x})$. The more $\mathbf{x}'$ is preferred over $\mathbf{x}$, the bigger the reward.

We define the model of preference using a Bernoulli likelihood, where $p(y = 1 \mid [\mathbf{x}, \mathbf{x}']) = \pi_f([\mathbf{x}, \mathbf{x}'])$ and $p(y = 0 \mid [\mathbf{x}, \mathbf{x}']) = \pi_f([\mathbf{x}', \mathbf{x}])$ for some inverse link function $\pi : \mathfrak{R} \times \mathfrak{R} \to [0, 1]$. π_f has the property that $\pi_f([\mathbf{x}', \mathbf{x}]) = 1 - \pi_f([\mathbf{x}, \mathbf{x}'])$. A natural choice for π_f is the logistic function

$$\pi_f([\mathbf{x}, \mathbf{x}']) = \sigma(f([\mathbf{x}, \mathbf{x}'])) = \frac{1}{1 + e^{-f([\mathbf{x}, \mathbf{x}'])}},$$

but others are possible. Therefore we have that for any duel $[\mathbf{x}, \mathbf{x}']$ in which $g(\mathbf{x}) \leq g(\mathbf{x}')$ it holds that $\pi_f([\mathbf{x}, \mathbf{x}']) \geq 0.5$. π_f is a preference function that maps each query $[\mathbf{x}, \mathbf{x}']$ to the probability of having a preference on the left input $\mathbf{x}$ over the right input $\mathbf{x}'$.

When we marginalize over the right input $\mathbf{x}'$ of f, the global minimum of f in $\mathcal{X}$ coincides with $\mathbf{x}_{\min}$. We also introduce the definition of the *Copeland score function* for a point $\mathbf{x}$ as

$$S(\mathbf{x}) = \text{Vol}(\mathcal{X})^{-1} \int_{\mathcal{X}} \mathbb{I}_{\{\pi_f([\mathbf{x}, \mathbf{x}']) \geq 0.5\}} d\mathbf{x}' \tag{4.45}$$

where $\text{Vol}(\mathcal{X}) = \int_{\mathcal{X}} d\mathbf{x}'$ is a normalizing constant that bounds $S(\mathbf{x})$ in the interval $[0, 1]$. If $\mathcal{X}$ is a finite set, the Copeland score is simply the proportion of duels that a certain element $\mathbf{x}$ will win with probability larger than 0.5. A soft variant we will use instead of the Copeland score is the *soft-Copeland score*, defined as

$$C(\mathbf{x}) = \text{Vol}(\mathcal{X})^{-1} \int_{\mathcal{X}} \pi_f([\mathbf{x}, \mathbf{x}'])\, d\mathbf{x}'$$

where the probability function π_f is integrated over $\mathcal{X}$. This score aims to capture the average probability of $\mathbf{x}$ being the winner of a duel.

We define the *Condorcet winner* $\mathbf{x}_c$ as the point with maximal soft-Copeland score. Note that this corresponds to the global minimum of f, since the defining integral takes maximum value for points $\mathbf{x} \in \mathcal{X}$ where $f([\mathbf{x}, \mathbf{x}']) = g(\mathbf{x}') - g(\mathbf{x}) > 0$ or all $\mathbf{x}'$, occurring only if $\mathbf{x}_c$ is a minimum of f. Therefore, if the preference function π_f can be learned by observing the results of duels then our optimization problem of finding the minimum of f can be solved by finding the Condorcet winner of the Copeland score.

Code: Copeland Score Visualization

```
#| autorun: true
# Visualize Copeland score for a 1D latent function
def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

# True latent function (to minimize)
def g(x):
    return (x - 0.3)**2 + 0.5 * np.sin(4 * x)

# Preference function: P(x beats x') = sigmoid(g(x') - g(x))
def preference_prob(x, xp):
    return sigmoid(g(xp) - g(x))

# Compute soft-Copeland score at each point
X = np.linspace(-1, 1, 100)
X_sample = np.linspace(-1, 1, 50)  # Points to compare against

copeland_scores = []
for x in X:
    # Monte Carlo estimate of soft-Copeland
    probs = [preference_prob(x, xp) for xp in X_sample]
    copeland_scores.append(np.mean(probs))
copeland_scores = np.array(copeland_scores)

# Find Condorcet winner (maximizer of Copeland score)
condorcet_idx = np.argmax(copeland_scores)
condorcet_x = X[condorcet_idx]

# Also find true minimizer of g
true_min_idx = np.argmin(g(X))
true_min_x = X[true_min_idx]

fig, axes = plt.subplots(1, 2, figsize=(6, 3))

# Left: Latent function g(x)
axes[0].plot(X, g(X), 'b-', linewidth=1.5, label='$g(x)$')
axes[0].axvline(true_min_x, color='green', linestyle='--', alpha=0.7,
 ↪  label=f'True min: {true_min_x:.2f}')
axes[0].scatter([true_min_x], [g(true_min_x)], color='green', s=80, zorder=5)
axes[0].set_xlabel('$x$')
axes[0].set_ylabel('$g(x)$')
axes[0].set_title('Latent Function (to minimize)')
axes[0].legend(fontsize=7)
axes[0].grid(True, alpha=0.3)

# Right: Soft-Copeland score
axes[1].plot(X, copeland_scores, 'r-', linewidth=1.5, label='Soft-Copeland
 ↪  $C(x)$')
axes[1].axvline(condorcet_x, color='purple', linestyle='--', alpha=0.7,
 ↪  label=f'Condorcet winner: {condorcet_x:.2f}')
axes[1].scatter([condorcet_x], [copeland_scores[condorcet_idx]], color='purple',
 ↪  s=80, zorder=5)
axes[1].set_xlabel('$x$')
```

Code: GP Preference Model for PBO

```
#| autorun: true
class GPPreferenceModel:
    """GP-based preference model for Preferential Bayesian Optimization."""

    def __init__(self, lengthscale=0.3, variance=1.0):
        self.lengthscale = lengthscale
        self.variance = variance
        self.X_duels = []  # [(x, x'), ...]
        self.y_duels = []  # [1 if x wins, 0 if x' wins]

    def kernel(self, X1, X2):
        """RBF kernel on duel space."""
        diff = X1[:, None] - X2[None, :]
        return self.variance * np.exp(-0.5 * diff**2 / self.lengthscale**2)

    def add_duel(self, x, xp, winner):
        """Add a duel observation. winner=1 means x wins."""
        self.X_duels.append((x, xp))
        self.y_duels.append(winner)

    def predict(self, x_test, xp_test):
        """Predict P(x beats x') using GP classification."""
        if len(self.X_duels) == 0:
            return 0.5, 0.5  # Prior: uniform

        # Represent duels as difference in 1D
        X_train = np.array([d[0] - d[1] for d in self.X_duels])
        y_train = np.array(self.y_duels)
        x_diff = x_test - xp_test

        # GP posterior (simplified Laplace)
        K = self.kernel(X_train, X_train) + 1e-4 * np.eye(len(X_train))
        k_star = self.kernel(np.array([x_diff]), X_train).flatten()
        k_ss = self.kernel(np.array([x_diff]), np.array([x_diff]))[0, 0]

        # Laplace approximation
        f = np.zeros(len(y_train))
        for _ in range(5):
            p = sigmoid(f)
            W = np.diag(p * (1 - p) + 1e-6)
            grad = y_train - p
            H = W + np.linalg.inv(K)
            f = f + np.linalg.solve(H, grad)

        # Posterior mean and variance
        K_inv = np.linalg.inv(K)
        mu = k_star @ K_inv @ f
        p = sigmoid(f)
        W = np.diag(p * (1 - p) + 1e-6)
        var = k_ss - k_star @ np.linalg.inv(K + np.linalg.inv(W)) @ k_star
        var = max(var, 1e-6)

        return sigmoid(mu), np.sqrt(var)
```

4.6.2 Acquisition Functions

We describe several acquisition functions for sequential learning of the Condorcet winner. Our dataset $\mathcal{D} = \{[\mathbf{x}_i, \mathbf{x}'_i], y_i\}_{i=1}^N$ represents the N duels that have been performed so far. We aim to define a sequential policy $\alpha([\mathbf{x}, \mathbf{x}']; \mathcal{D}_j, \theta)$ for querying duels, where θ is a vector of model hyper-parameters, in order to find the minimum of the latent function g as quickly as possible. Using Gaussian processes (GP) for classification with our dataset $\mathcal{D}$ allows us to perform inference over f and π_f.

Pure Exploration

The output variable $y_\star$ of a prediction follows a Bernoulli distribution with probability given by the preference function π_f. To carry out exploration as a policy, one method is to search for the duel where GP is most uncertain about the probability of the outcome (has the highest variance of $\sigma(f_\star)$), which is the result of transforming out epistemic uncertainty about f, modeled by a GP, through the logistic function. The first order moment of this distribution coincides with the expectation of $y_\star$ but its variance is

$$\begin{aligned}\mathbb{V}[\sigma(f_\star)] &= \int (\sigma(f_\star) - \mathbb{E}[\sigma(f_\star)])^2 \, p(f_\star \mid \mathcal{D}, [\mathbf{x}, \mathbf{x}']) \, df_\star \\ &= \int \sigma(f_\star)^2 \, p(f_\star \mid \mathcal{D}, [\mathbf{x}, \mathbf{x}']) \, df_\star - \mathbb{E}[\sigma(f_\star)]^2\end{aligned} \tag{4.46}$$

which explicitly takes into account the uncertainty over f. Hence, pure exploration of duels space can be carried out by maximizing

$$\alpha_{\text{PE}}([\mathbf{x}, \mathbf{x}'] \mid \mathcal{D}_j) = \mathbb{V}\left[\sigma(f_\star) \, | [\mathbf{x}_\star, \mathbf{x}'_\star] | \, \mathcal{D}_j\right]. \tag{4.47}$$

Note that in this case, duels that have been already visited will have a lower chance of being visited again even in cases in which the objective takes similar values in both players. In practice, this acquisition functions requires computation of an intractable integral, that we approximate using Monte-Carlo.

Principled Optimistic Preferential Bayesian Optimization (POP-BO)

In a slightly modified problem setup (Xu et al. 2024), the algorithm tries to solve for the MLE $\hat{g}$ and its confidence set $\mathcal{B}_g$ where g is the ground truth black-box function. Assumptions include that g is a member of a reproducing kernel Hilbert space (RKHS) $\mathcal{H}_k$ for some kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, and $\|g\|_k \leq B$ so that $\mathcal{B}_g = \{\tilde{g} \in \mathcal{H}_k \mid \|\tilde{g}\|_k \leq B\}$. Similarly defining $f([\mathbf{x}, \mathbf{x}']) = g(\mathbf{x}') - g(\mathbf{x})$, we model the preference function with a Bernoulli distribution as in Equation [eq:bernoulli_pref] and also assume that probabilities follow the Bradley-Terry

model, i.e.

$$\pi_f\left([\mathbf{x}, \mathbf{x}']\right) = \sigma\left(f\left([\mathbf{x}, \mathbf{x}']\right)\right) = \frac{e^{g(\mathbf{x})}}{e^{g(\mathbf{x})} + e^{g(\mathbf{x}')}} \tag{4.48}$$

The update rule for MLE $\hat{g}$ is (equation 8,6,5)

$$\begin{aligned}
\hat{g}_t^{\text{MLE}} &:= \arg\max_{\tilde{g} \in \mathcal{B}_g^t} \ell_t(\tilde{g}) \\
\ell_t(\tilde{g}) &:= \log \prod_{\tau=1}^{t} y_\tau \pi_{\tilde{f}}([\mathbf{x} \ , \mathbf{x}' \,]) + (1 - y_\tau)\left(1 - \pi_{\tilde{f}}([\mathbf{x} \ , \mathbf{x}' \,])\right) \\
&= \sum_{\tau=1}^{t} \log\left(\frac{e^{\tilde{g}(\mathbf{x} \ )} y_\tau + e^{\tilde{g}(\mathbf{x}')}\left(1 - y_\tau\right)}{e^{\tilde{g}(\mathbf{x} \ )} + e^{\tilde{g}(\mathbf{x}')}}\right) \\
&= \sum_{\tau=1}^{t} \left(\tilde{g}(\mathbf{x} \ ) y_\tau + \tilde{g}(\mathbf{x}' \,)\left(1 - y_\tau\right)\right) - \sum_{\tau=1}^{t} \log\left(e^{\tilde{g}(\mathbf{x} \ )} + e^{\tilde{g}(\mathbf{x}')}\right)
\end{aligned} \tag{4.49}$$

(Eq 22 shows how to represent this as a convex optimisation problem so that it can be solved)

The update rule for the confidence set $\mathcal{B}_f^{t+1}$ is, (eq 9, 10?)

$$\begin{aligned}
&\forall \epsilon, \delta > 0 \\
&\mathcal{B}_g^{t+1} := \left\{\tilde{g} \in \mathcal{B}_g \mid \ell_t(\tilde{g}) \geq \ell_t\left(\hat{g}_t^{\text{MLE}}\right) - \beta_1(\epsilon, \delta, t)\right\}
\end{aligned} \tag{4.50}$$

where

$$\beta_1(\epsilon, \delta, t) := \sqrt{32tB^2 \log \frac{\pi^2 t^2 \mathcal{N}\left(\mathcal{B}_f, \epsilon, \|\cdot\|_\infty\right)}{6\delta}} + C_L \epsilon t = \mathcal{O}\left(\sqrt{t \log \frac{t\mathcal{N}\left(\mathcal{B}_f, \epsilon, \|\cdot\|_\infty\right)}{\delta}} + \epsilon t\right), \tag{4.51}$$

with C_L a constant independent of δ, t and ϵ. ϵ is typically chosen to be $1/T$, where T is the running horizon of the algorithm. This satisfies the theorem that,

$$\mathbb{P}\left(g \in \mathcal{B}_g^{t+1}, \forall t \geq 1\right) \geq 1 - \delta. \tag{4.52}$$

Intuitively, the confidence set $\mathcal{B}_g^{t+1}$ includes the functions with the log-likelihood value that is only 'a little worse' than the maximum likelihood estimator, and the theorem states that $\mathcal{B}_g^{t+1}$ contains the ground-truth function g with high probability.

Inner level optimization in Line 4 of the algorithm can also be represented as a convex optimisation problem so that it can be solved, Eq 24, 25. The outer optimisation can be solved using grid search or Eq 26 for medium size problems.

Given the initial point $\mathbf{x_0} \in \mathcal{X}$ and set $\mathcal{B}_g^1 = \mathcal{B}_g$ Set the reference point $\mathbf{x_t'} = \mathbf{x_{t-1}}$ Compute $\mathbf{x_t} \in \arg\max_{\mathbf{x} \in \mathcal{X}} \max_{\tilde{g} \in \mathcal{B}_g^t} (\tilde{g}(\mathbf{x}) - \tilde{g}(\mathbf{x_t'}))$, with the inner optimal function denoted as $\tilde{g}_t$ Obtain

the output of the duel y_t and append the new data point to $\mathcal{D}_t$ Update the maximum likelihood estimator $\hat{g}_t^{\text{MLE}}$ and the posterior confidence set $\mathcal{B}_g^{t+1}$.

qEUBO: Decision-Theoretic EUBO

qEUBO (Astudillo et al. 2023) derives an acquisition function that extends duels to $q > 2$ options which we call *queries*. Let $X = \left(\mathbf{x_1}, \ldots, \mathbf{x_q}\right) \in \mathcal{X}^q$ denote a query containing two points or more, and let $g : \mathcal{X} \to \Re$ be the latent preference function. Then after n user queries, we define the *expected utility of the best option* (qEUBO) as

$$\text{qEUBO}_n(X) = \mathbb{E}_n \left[\max \left\{g\left(x_1\right), \ldots, g\left(x_q\right)\right\}\right]. \tag{4.53}$$

We now show that qEUBO is one-step Bayes optimal, meaning that each step chooses the query that maximises the expected utility received by the human. For a query $X \in \mathcal{X}^q$, let

$$V_n(X) = \mathbb{E}_n \left[\max_{x \in \mathbb{X}} \mathbb{E}_{n+1}[g(x)] \mid X_{n+1} = X\right]. \tag{4.54}$$

Then V_n defines the expected utility received if an additional query $X_{n+1} = X$ is performed, and maximizing V_n is one-step Bayes optimal. Since $\max_{x \in \mathbb{X}} \mathbb{E}_n[f(x)]$ does not depend on X_{n+1}, we can also equivalently maximize

$$\mathbb{E}_n \left[\max_{x \in \mathbb{X}} \mathbb{E}_{n+1}[g(x)] - \max_{x \in \mathbb{X}} \mathbb{E}_n[g(x)] \mid X_{n+1} = X\right], \tag{4.55}$$

which takes the same form as the knowledge gradient acquisition function (Wu and Frazier 2018) in standard Bayesian optimization.

V_n involves a nested stochastic optimization task, while qEUBO is a much simpler policy. When human responses are noise-free, we are able to use qEUBO as a sufficient policy due to the following theorem:

$$\underset{X \in \mathbb{X}^q}{\text{argmax}} \text{qEUBO}_n(X) \subseteq \underset{X \in \mathbb{X}^q}{\text{argmax}} V_n(X). \tag{4.56}$$

Proof sketch. The proof establishes two key inequalities. First, the one-step value $V_n(X)$ is bounded above by $\text{qEUBO}_n(X^+(X))$, where $X^+(X)$ contains the best post-observation recommendations—because selecting the best option from a query can only be as good as already knowing the best option in the pool. Second, $\text{qEUBO}_n(X) \leq V_n(X)$ because the user-selected option from the query may not be the global optimum. Combining these via a proof by contradiction shows that any maximizer of qEUBO is also a maximizer of the one-step Bayes-optimal value function V_n. See Astudillo et al. (2023) for the full proof. □

Therefore a sufficient condition for following one-step Bayes optimality is by maximizing qEUBO_n.

In experiments comparing qEUBO to other state-of-the-art acquisition functions, qEUBO consistently outperformed on most problems and was closely followed by qEI and qTS. These results also extended to experiments with multiple options when $q > 2$. In fact, there is faster convergence in regret when using more options in human queries. The regret analysis is presented in detail below.

qEI: Batch Expected Improvement

$$\begin{aligned} \text{qEI} =& \mathbb{E}_{\mathbf{y}}\left[\left(\max_{i\in[1,\ldots,q]}\left(\mu_{\min}-y_i\right)\right)_+\right] \\ =& \sum_{i=1}^{q}\mathbb{E}_{\mathbf{y}}\left(\mu_{\min}-y_i \mid y_i\leq\mu_{\min}, y_i\leq y_j \forall j\neq i\right) \\ & p\left(y_i\leq\mu_{\min}, y_i\leq y_j \forall j\neq i\right). \end{aligned} \tag{4.57}$$

qTS: Batch Thompson Sampling

Initial data $\mathcal{D}_{\mathcal{J}(1)} = \{(\mathbf{x}_i, y_i)\}_{i\in\mathcal{J}(1)}$ Compute current posterior $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Sample θ from $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Select $k \leftarrow \arg\max_{j\notin\mathcal{J}(t)} \mathbb{E}[y_j \mid \mathbf{x}_j, \theta]$ Collect y_k by evaluating f at $\mathbf{x}_k$ $\mathcal{D}_{\mathcal{J}(t+1)} \leftarrow \mathcal{D}_{\mathcal{J}(t)} \cup \{(\mathbf{x}_k, y_k)\}$

Initial data $\mathcal{D}_{\mathcal{J}(1)} = \{\mathbf{x}_i, y_i\}_{i\in\mathcal{J}(1)}$, batch size S Compute current posterior $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Sample θ from $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Select $k(s) \leftarrow \arg\max_{j\notin\mathcal{J}(t)} \mathbb{E}[y_j \mid \mathbf{x}_j, \theta]$ $\mathcal{D}_{\mathcal{J}(t+1)} = \mathcal{D}_{\mathcal{J}(t)} \cup \{\mathbf{x}_{k(s)}, y_{k(s)}\}_{s=1}^{S}$

4.6.3 Regret Analysis

qEUBO Regret

With the definition of Bayesian simple regret, we have that qEUBO converges to zero at a rate of $o(1/n)$, i.e.

$$\mathbb{E}\left[f\left(x^*\right) - f\left(\hat{x}_n^*\right)\right] = o(1/n)$$

where $x^* = \text{argmax}_{x\in\mathbb{X}} f(x)$ and $\hat{x}_n^* \in \text{argmax}_{x\in\mathbb{X}} \mathbb{E}_n[f(x)]$.

This theorem holds under the following assumptions:

1. f **is injective** $\mathbf{P}(f(x) = f(y)) = 0$ for any $x, y \in \mathbb{X}$ with $x \neq y$.
2. f **represents the preferred option** $\exists a > 1/2$ s.t. $\mathbf{P}\left(r(X) \in \text{argmax}_{i=1,\ldots,2} f\left(x_i\right) \mid f(X)\right) \geq a \forall\, X = (x_1, x_2) \in \mathbb{X}^2$ with $x_1 \neq x_2$ almost surely under the prior on f.

3. **Expected difference in utility is proportional to probability of greater utility** $\exists \Delta \geq \delta > 0$ s.t. $\forall \mathcal{D}^{(n)}and\forall x, y \in \mathbb{X}$ (potentially depending on $\mathcal{D}^{(n)}$),

$$\delta \mathbf{P}^{(n)}(f(x) > f(y)) \leq \mathbb{E}^{(n)}\left[\{f(x) - f(y)\}^{+}\right] \leq \Delta \mathbf{P}^{(n)}(f(x) > f(y)) \tag{4.58}$$

almost surely under the prior on f.

Further lemmas leading to a proof of Theorem [th:quebo_regret] is given in (Astudillo et al. 2023) Section B.

qEI Regret

The following theorem shows that, under the same assumptions used for qEUBO regret, simple regret of qEI can fail to converge to 0.

There exists a problem instance (i.e., $\mathbb{X}$ and Bayesian prior distribution over f) satisfying the assumptions described in Theorem [th:quebo_regret] such that if the sequence of queries is chosen by maximizing qEI, then $\mathbb{E}\left[f\left(x^{*}\right) - f\left(\hat{x}_{n}^{*}\right)\right] \geq R$ for all n, for a constant $R > 0$.

Proof sketch. The counterexample constructs four functions on $\mathcal{X} = \{1, 2, 3, 4\}$ with a carefully chosen prior. The key insight is that qEI optimizes improvement over the *current best estimate*, which in this construction is always $x = 2$ (with $f(2) = 0$). Since comparing $x = 3$ vs. $x = 4$ always yields the highest expected improvement over $f(2) = 0$, qEI perpetually selects the query $(3, 4)$—never querying the pairs that would reveal whether the true optimum lies at $x = 3$ or $x = 4$. One can show by induction that this trapping behavior persists for all n, yielding constant Bayesian simple regret $R = p > 0$. In contrast, qEUBO avoids this trap because it maximizes the expected utility of the *best option shown*, which naturally encourages diverse queries. See Astudillo et al. (2023) Section B for the full induction argument. □

POP-BO Regret

POP-BO achieves provable regret guarantees that depend on the choice of kernel. The cumulative regret bound involves the *maximum information gain* $\gamma_T^{ff'}$, which measures the complexity of the function class defined by the kernel.

With probability at least $1 - \delta$, the cumulative regret of POP-BO satisfies

$$R_T = \mathcal{O}\left(\sqrt{\beta_T \gamma_T^{ff'} T}\right), \tag{4.59}$$

and the simple regret of the reported solution converges as

$$f\left(x^{\star}\right) - f\left(x_{t^{\star}}\right) \leq \mathcal{O}\left(\frac{\sqrt{\beta_T \gamma_T^{ff'}}}{\sqrt{T}}\right). \tag{4.60}$$

The kernel-specific rates reveal how the smoothness of the preference function affects convergence:

For POP-BO with $\epsilon = 1/T$:

1. **Linear kernel** $k(x, y) = \langle x, y \rangle$: $R_T = \mathcal{O}\left(T^{3/4}(\log T)^{3/4}\right)$
2. **Squared exponential kernel**: $R_T = \mathcal{O}\left(T^{3/4}(\log T)^{3/4(d+1)}\right)$
3. **Matérn kernel** (smoothness ν): $R_T = \mathcal{O}\left(T^{3/4}(\log T)^{3/4} T^{\frac{d}{\nu}\left(\frac{1}{4} + \frac{d+1}{4+2(d+1)^d/\nu}\right)}\right)$

The rates share a $T^{3/4}$ base rate, with kernel-dependent logarithmic or polynomial corrections. Smoother kernels (larger ν in Matérn, or SE which corresponds to $\nu \to \infty$) yield tighter bounds, reflecting the intuition that smoother preference functions are easier to optimize from pairwise comparisons. Compared to standard BO (which achieves $T^{1/2}$ rates), the extra $T^{1/4}$ factor is the price of learning from preference feedback rather than direct function evaluations.

Code: Full PBO Simulation

This simulation demonstrates Preferential Bayesian Optimization finding the optimum of a latent function through pairwise comparisons.

```
#| autorun: true
# Full PBO simulation
np.random.seed(42)

# True latent function (unknown to algorithm)
def true_g(x):
    return (x - 0.2)**2 + 0.3 * np.sin(5 * x)

# True preference probability
def true_pref(x, xp):
    return sigmoid(true_g(xp) - true_g(x))
```

4.7 Reinforcement Learning Foundations for Preference Optimization (Optional)

> **Note**
>
> This section is optional and covers reinforcement learning (RL) foundations that underlie modern preference optimization methods like RLHF, PPO, and GRPO. It provides the formal framework for CIRL (next section) and connects the bandit methods from earlier sections to the full sequential setting used in language model alignment. Readers familiar with RL may wish to skim Sections 4.6.1–4.6.2 and focus on Sections 4.6.3–4.6.5, which develop the policy gradient theorem and GRPO in the preference optimization context.

The bandit and Bayesian optimization methods developed so far operate in **single-step** or **myopic** settings: at each round, the learner selects an arm or a query pair, observes feedback, and updates its model. But the RLHF pipeline introduced in Chapter 1 and the CIRL framework in the next section both require reasoning over **sequences** of decisions, where early actions influence future states.

This section develops the RL machinery needed for these settings, culminating in **GRPO** (Group Relative Policy Optimization)—a modern method that directly connects policy gradient optimization to the Bradley-Terry preference model from Chapter 1. The progression from bandits to full RL mirrors this chapter's arc from simple to complex decision-making under uncertainty.

4.7.1 Markov Decision Processes

The **Markov Decision Process** (MDP) generalizes the bandit setting by introducing *states* that evolve over time based on the learner's actions.

> **Definition: Markov Decision Process (MDP)**
>
> An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, \rho_0)$ where:
>
> - $\mathcal{S}$: state space
> - $\mathcal{A}$: action space
> - $P(s' \mid s, a)$: transition probability from state s to s' under action a
> - $R(s, a)$: reward function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
> - $\gamma \in [0, 1]$: discount factor
> - ρ_0: initial state distribution

A **policy** $\pi_\theta(a \mid s)$ maps states to distributions over actions, parameterized by θ. An agent following policy π generates a **trajectory**:

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T), \quad s_0 \sim \rho_0,\ a_t \sim \pi(\cdot \mid s_t),\ s_{t+1} \sim P(\cdot \mid s_t, a_t) \tag{4.61}$$

The **RL objective** is to find the policy that maximizes expected cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \tag{4.62}$$

When $T = 0$ (a single step with no state transitions), the MDP reduces to the **bandit** setting from earlier sections. The RL objective then reduces to maximizing expected immediate reward $\mathbb{E}_{a \sim \pi_\theta}[R(a)]$, recovering the reward maximization problem from .

EXAMPLE: LLM GENERATION AS AN MDP

Autoregressive text generation fits naturally into the MDP framework:

- **State** s_t: the prompt x concatenated with tokens generated so far, $(x, y_1, \ldots, y_t)$
- **Action** a_t: the next token y_{t+1} from the vocabulary $\mathcal{V}$
- **Transition**: deterministic—the new state appends the chosen token
- **Reward**: $R(s_t, a_t) = 0$ for all intermediate steps; at the terminal step T, the reward is $r_\phi(x, y)$ from a learned reward model (e.g., trained on Bradley-Terry preferences as in Chapter 1)
- **Discount**: $\gamma = 1$ (finite horizon, all tokens matter equally)

The policy $\pi_\theta(a_t \mid s_t)$ is the language model itself, producing a distribution over the next token given the context.

The next section introduces CIRL, which extends this single-agent MDP to a **two-player cooperative game** where a human and robot jointly optimize a shared reward.

4.7.2 Value Functions and the Advantage

To understand *how much better* one action is compared to others, we define three related quantities.

The **state-value function** measures the expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s\right] \tag{4.63}$$

The **action-value function** additionally conditions on taking a specific first action:

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a\right] \tag{4.64}$$

The **advantage function** is the difference:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{4.65}$$

The advantage tells us how much better action a is compared to the *average* action under the current policy at state s. By definition, $\mathbb{E}_{a\sim\pi(\cdot|s)}[A^\pi(s,a)] = 0$—some actions have positive advantage (better than average) and others negative. This quantity will reappear as the key ingredient in GRPO's group-relative baseline.

These quantities satisfy the **Bellman equation**, which recursively relates the value of a state to the values of its successors:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[R(s,a) + \gamma\, \mathbb{E}_{s'\sim P(\cdot|s,a)}[V^\pi(s')]\right] \tag{4.66}$$

4.7.3 The Policy Gradient Theorem

We now derive the central result that enables gradient-based policy optimization. The challenge is that the RL objective depends on θ both through the policy π_θ (which determines which actions are taken) and through the resulting trajectory distribution (which determines which states are visited). The **policy gradient theorem** shows that we can compute gradients using only the policy's log-probabilities, without differentiating through the environment dynamics.

The score function trick. For any parameterized distribution $\pi_\theta(a \mid s)$, we can write:

$$\nabla_\theta \pi_\theta(a \mid s) = \pi_\theta(a \mid s)\, \nabla_\theta \log \pi_\theta(a \mid s) \tag{4.67}$$

This identity converts a gradient of a probability into an expectation-friendly form. The term $\nabla_\theta \log \pi_\theta(a \mid s)$ is called the **score function**—the same quantity that appears in the Fisher information matrix (Chapter 3), though here it serves a different purpose: instead of measuring information content, it tells us how to adjust the policy to increase reward.

Trajectory probability. The probability of a trajectory under policy π_θ factors as:

$$p(\tau \mid \theta) = \rho_0(s_0) \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t) \tag{4.68}$$

Taking the log-gradient, the transition terms $P(s_{t+1} \mid s_t, a_t)$ vanish because they do not depend on θ:

$$\nabla_\theta \log p(\tau \mid \theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \tag{4.69}$$

This is the key insight: **the environment dynamics drop out of the gradient**, so we can estimate policy gradients without knowing or differentiating through the transition function P.

THEOREM: POLICY GRADIENT

For a parameterized policy π_θ with RL objective $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\gamma^t R(s_t, a_t)]$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\cdot G_t\right] \tag{4.70}$$

where $G_t = \sum_{t'=t}^{T}\gamma^{t'-t}R(s_{t'}, a_{t'})$ is the **return** (cumulative discounted reward) from time step t onward.

Proof sketch. Write $J(\theta) = \mathbb{E}_{\tau\sim p(\cdot|\theta)}[R(\tau)]$ where $R(\tau) = \sum_t \gamma^t R(s_t, a_t)$. Apply the score function trick to the trajectory distribution: $\nabla_\theta J = \mathbb{E}_\tau[R(\tau)\nabla_\theta \log p(\tau|\theta)]$. Substitute . Finally, note that actions at time t cannot influence rewards at earlier times, so we can replace $R(\tau)$ with the future return G_t for each term in the sum. □

4.7.4 REINFORCE and Variance Reduction

The policy gradient theorem immediately yields the simplest policy gradient algorithm.

ALGORITHM: REINFORCE (WILLIAMS 1992)

Input: Parameterized policy π_θ, learning rate α

Repeat:

1. Sample trajectory $\tau = (s_0, a_0, \ldots, s_T, a_T) \sim \pi_\theta$
2. For each time step $t = 0, \ldots, T$:
   - Compute return $G_t = \sum_{t'=t}^{T}\gamma^{t'-t}R(s_{t'}, a_{t'})$
3. Update: $\theta \leftarrow \theta + \alpha\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\cdot G_t$

REINFORCE is an unbiased estimator of the policy gradient, but it suffers from **high variance**: a single trajectory can produce wildly different gradient estimates depending on the stochastic outcomes. This motivates **variance reduction** through baselines.

Baseline subtraction. We can subtract any function $b(s_t)$ that depends only on the state (not the action) from the return without introducing bias:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\,(G_t - b(s_t))\right] \tag{4.71}$$

Why is this unbiased? For any state-dependent baseline $b(s)$:

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a \mid s)\cdot b(s)\right] = b(s)\cdot\nabla_\theta\sum_a \pi_\theta(a \mid s) = b(s)\cdot\nabla_\theta 1 = 0 \tag{4.72}$$

The probabilities sum to 1 regardless of θ, so the gradient of that sum is zero.

The variance-minimizing baseline turns out to be $b^*(s_t) = V^\pi(s_t)$, which reduces the gradient to use the **advantage** $A^\pi(s_t, a_t) = G_t - V^\pi(s_t)$. Intuitively, we only reinforce actions that are *better than average*—actions with positive advantage get upweighted, while those with negative advantage get downweighted.

> **CAUTION**
>
> In practice, $V^\pi(s)$ is unknown and must itself be estimated, typically by training a separate neural network (the **critic** or **value network**) alongside the policy. This adds computational cost and can introduce its own approximation errors. GRPO, discussed next, avoids this entirely.

4.7.5 From RLHF to GRPO

We now connect the policy gradient framework to the **RLHF pipeline** introduced in Chapter 1 and develop GRPO as a modern, critic-free alternative to PPO.

The RLHF objective. Recall from Chapter 1 that RLHF trains a reward model $r_\phi(x, y)$ on human preference data using the Bradley-Terry likelihood, then optimizes the policy to maximize reward while staying close to a reference policy π_{ref}:

$$J_{\text{RLHF}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[r_\phi(x, y)\right] - \beta\,\mathrm{KL}\left[\pi_\theta(\cdot|x) \,\|\, \pi_{\text{ref}}(\cdot|x)\right] \tag{4.73}$$

The KL penalty prevents **reward hacking**: without it, the policy could exploit imperfections in the learned reward model r_ϕ to achieve high reward scores that do not correspond to genuinely preferred outputs (see Chapter 7 for empirical discussion).

PPO. Proximal Policy Optimization (Schulman et al. 2017) applies the policy gradient theorem to the RLHF objective using a **clipped surrogate** that prevents excessively large policy updates. PPO requires a learned value function $V_\psi(s)$ (the critic) to estimate advantages. Training this critic adds computational cost—roughly doubling the model parameters in the LLM setting, since the critic is typically the same size as the policy. Trust Region Policy Optimization [TRPO; Schulman et al. (2015)] preceded PPO with a similar motivation but used a hard KL constraint instead of clipping.

GRPO: Group Relative Policy Optimization. GRPO (Shao et al. 2024) eliminates the critic by replacing the learned value baseline with a **group-relative** baseline computed from multiple sampled outputs. The key idea is conceptually parallel to the dueling bandit paradigm from earlier in this chapter: rather than estimating absolute values (which requires a critic), we compare outputs *within a group* to determine which are relatively better.

ALGORITHM: GROUP RELATIVE POLICY OPTIMIZATION (GRPO)

Input: Policy π_θ, reference policy π_{ref}, reward model r_ϕ, group size G, clip range ϵ, KL weight β

For each prompt $x \sim \mathcal{D}$:

1. **Sample** a group of outputs: $\{y_1, \ldots, y_G\} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$
2. **Score** each output: $r_i = r_\phi(x, y_i)$ for $i = 1, \ldots, G$
3. **Normalize** to get group-relative advantages:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^G)}{\text{std}(\{r_j\}_{j=1}^G)} \tag{4.74}$$

4. **Update** the policy by maximizing:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \frac{1}{G}\sum_{i=1}^{G} \min\Big(\rho_i \hat{A}_i,\ \text{clip}(\rho_i, 1{-}\epsilon, 1{+}\epsilon)\,\hat{A}_i\Big) - \beta\,\text{KL}[\pi_\theta \| \pi_{\text{ref}}] \tag{4.75}$$

where $\rho_i = \pi_\theta(y_i \mid x) \,/\, \pi_{\theta_{\text{old}}}(y_i \mid x)$ is the importance ratio.

The z-score normalization in plays the same role as pairwise comparisons in dueling bandits: it tells us which outputs are *relatively* better within the group, without requiring an absolute value estimate. When the group mean is high (all outputs are good), only the *best* outputs receive positive advantage; when the group mean is low, even mediocre outputs can be positively reinforced.

KEY INSIGHT: GRPO AND DUELING BANDITS

GRPO's group-relative baseline is the RL analog of the dueling bandit paradigm. In dueling bandits, we never observe absolute arm rewards—only pairwise comparisons. Similarly, GRPO never estimates absolute state values—only relative quality within a sampled group. This connection runs deeper: the Bradley-Terry reward model r_ϕ that scores GRPO's outputs is the same preference model that defines the dueling bandit's comparison probabilities. The circle from preference models (Chapter 1) through sequential decision-making (this chapter) back to policy optimization closes here.

Connection to DPO. When $G = 2$ and the reward comes from a Bradley-Terry model, GRPO's advantage reduces to $\hat{A}_1 = -\hat{A}_2 \propto r_\phi(x, y_1) - r_\phi(x, y_2)$. This reward difference is exactly the quantity that determines preference probabilities in the Bradley-Terry model: $p(y_1 \succ y_2 \mid x) = \sigma(r_\phi(x, y_1) - r_\phi(x, y_2))$. DPO optimizes this signal directly on a fixed offline dataset, while GRPO generates fresh samples during training and generalizes to groups larger than 2. The online sampling allows GRPO to explore the policy's current output distribution, while DPO is limited to the distribution that generated the training data.

Code: REINFORCE vs. GRPO on a Preference Bandit

```
#| autorun: true
# Compare REINFORCE and GRPO on a toy preference optimization problem
np.random.seed(42)

# Setup: 5 "responses" with hidden Bradley-Terry utilities
n_actions = 5
true_utilities = np.array([0.5, 1.0, 2.0, 0.3, 1.5])  # Action 2 is best
temperature = 1.0

def reward_model(action):
    """Simulate a Bradley-Terry reward model with noise."""
    return true_utilities[action] + np.random.normal(0, 0.5)

def softmax_policy(logits):
    """Convert logits to action probabilities."""
    logits = logits - np.max(logits)  # numerical stability
    exp_logits = np.exp(logits)
    return exp_logits / np.sum(exp_logits)

```

4.7.6 Connections

The progression from bandits (single-step) to RL (sequential) to preference-based RL (RLHF, GRPO) mirrors this chapter's arc from simple to complex decision-making. The **policy gradient theorem** () provides the mathematical bridge: it tells us how to optimize *any* parameterized policy using only sampled trajectories and rewards. When those rewards come from a learned preference model (Bradley-Terry), we recover the modern alignment pipeline.

- **Forward to CIRL** (next section): The MDP framework developed here provides the formal setting for CIRL, where the single-agent MDP becomes a two-player cooperative game with shared rewards.
- **Forward to Chapter 7**: Chapter 7 discusses the practical lessons from deploying RLHF and GRPO at scale, including reward hacking and the gap between learned and true rewards.
- **Back to Chapter 3**: The log-linear policy assumption used in ADPO—$\pi(y \mid x; \theta) \propto \exp(\phi(x, y)^\top \theta)$—is a special case of the parameterized policy π_θ, specifically the softmax policy class common in RL.

Key Takeaway: From Bandits to RL to Preferences

GRPO further simplifies the RL pipeline by replacing the value function with group-relative comparisons, connecting RL optimization back to the pairwise comparison paradigm that runs throughout this book. The key equations—Bradley-Terry preference probabilities (Chapter 1), Fisher information and score functions (Chapter 3), and the policy gradient theorem (this section)—all share the same mathematical core: the log-derivative of a parameterized probability model.

4.8 Human-Agent Cooperation and CIRL

The algorithms discussed so far—Thompson Sampling, dueling bandits, Preferential Bayesian Optimization, and the RL methods in the preceding section—all share a common assumption: the human is a **passive oracle** who responds to queries but does not actively shape the learning process. In practice, this assumption often breaks down. A user choosing between LLM responses may deliberately select the more challenging example to *teach* the system. A patient in a clinical trial may modify their behavior based on what they believe the system is learning. Humans are **active agents** whose feedback depends on their beliefs about how it will be used.

This section introduces **Cooperative Inverse Reinforcement Learning (CIRL)**, a framework that models human and agent as jointly optimizing a shared objective. Where Thompson Sampling optimizes reward under uncertainty about the *reward function*, CIRL optimizes under uncertainty about the *human's preferences*—and crucially, the human knows this and can act to reduce that uncertainty.

4.8.1 The CIRL Framework

In standard Inverse Reinforcement Learning (IRL), the agent learns a reward function from observing an expert's behavior, then optimizes it. The expert demonstrates; the agent imitates. CIRL extends this to a **two-player cooperative game**:

- **Human H**: Knows the true reward parameters $\theta \in \Theta$
- **Robot R**: Does not know θ, but must act to maximize reward
- **Shared payoff**: Both players receive $R(s, a_H, a_R; \theta)$—their interests are aligned

The key insight is that the human's optimal policy is not just demonstrating good behavior, but **teaching** the robot about θ. Sometimes instructive actions (that reveal information) dominate expert demonstrations (that maximize immediate reward).

CIRL FORMAL DEFINITION

A CIRL game extends the single-agent MDP () to a two-player cooperative setting:

- State space $\mathcal{S}$
- Action spaces $\mathcal{A}_H$ (human) and $\mathcal{A}_R$ (robot)
- Transition function $T(s'|s, a_H, a_R)$
- Reward function $R(s, a_H, a_R; \theta)$ parameterized by θ
- Prior $P_0(\theta)$ over reward parameters
- Discount factor $\gamma \in (0, 1)$

At each time t, both observe $(s_t, a_H^{t-1}, a_R^{t-1})$. Only H observes θ. The robot maintains a belief $b_R^t(\theta)$ updated via Bayes' rule. The key difference from the standard MDP is that the reward parameter θ is observed only by the human, making this a game of **asymmetric information**.

4.8.2 Teaching vs. Demonstrating

A central result in CIRL is that **expert demonstrations can be suboptimal** for joint value. Consider:

- **Demonstration-by-expert**: Human maximizes task reward at each step
- **Instructive teaching**: Human sometimes sacrifices immediate reward to shape robot's belief

When the robot is uncertain, instructive actions that reveal θ can lead to higher cumulative reward than always acting optimally given θ.

Code: Teaching vs Demonstrating Simulation

```
#| autorun: true
# Simplified CIRL simulation: teaching vs demonstrating
np.random.seed(42)

# Two possible reward types: theta=0 (prefers action 0) or theta=1 (prefers
↪  action 1)
# Expert demonstration: always take optimal action for true theta
# Instructive teaching: sometimes take suboptimal action to reveal theta

def simulate_cirl(teaching=False, T=20):
    """Simulate CIRL interaction."""
    true_theta = np.random.randint(2)  # True reward type
    robot_belief = 0.5  # P(theta=1)

    rewards = []
    beliefs = [robot_belief]

    for t in range(T):
        if teaching and t < 3:
            # Instructive: exaggerate preference to teach robot
            human_action = true_theta
        else:
            # Demonstration: take optimal action (same as teaching when certain)
            human_action = true_theta

        # Robot chooses based on belief
        if robot_belief > 0.5:
            robot_action = 1
        elif robot_belief < 0.5:
            robot_action = 0
        else:
            robot_action = np.random.randint(2)

        # Reward: both get reward if robot matches true theta
        reward = 1.0 if robot_action == true_theta else 0.0
        rewards.append(reward)

        # Robot updates belief (simplified Bayesian update)
        # Observing human action provides signal about theta
        likelihood_1 = 0.9 if human_action == 1 else 0.1  # P(action | theta=1)
        likelihood_0 = 0.9 if human_action == 0 else 0.1  # P(action | theta=0)
        posterior_1 = robot_belief * likelihood_1
        posterior_0 = (1 - robot_belief) * likelihood_0
        robot_belief = posterior_1 / (posterior_1 + posterior_0 + 1e-10)
        beliefs.append(robot_belief)

    return np.cumsum(rewards), beliefs

# Compare strategies over multiple trials
n_trials = 50
demo_rewards = []
teach_rewards = []

```

4.8.3 Choice Architecture

As designers of preference learning systems, we are **choice architects**: the options we present and how we present them influence human behavior. Research in behavioral economics documents several effects:

- **Order effects**: Items presented first or last receive disproportionate attention
- **Framing effects**: How options are described changes perceived value
- **Default effects**: Pre-selected options are chosen more often

These effects are typically not captured in standard choice models (which assume IIA or similar properties). Responsible deployment of preference learning systems requires awareness of these biases.

Key Takeaway: Beyond Passive Oracles

The transition from measurement to decision-making revealed that uncertainty should be managed, not just reduced. The transition from passive oracles to active humans reveals that **humans themselves are strategic agents** whose behavior depends on their beliefs about the system. CIRL provides a principled framework for this cooperative setting.

Assumption: Approximately Rational Humans

CIRL's cooperative framework assumes the human acts **approximately optimally** given their beliefs about the robot's learning process. If the human is systematically biased—exhibiting anchoring effects, framing dependence, or status quo bias—the robot may learn incorrect preferences from the human's demonstrations. For example, a human who consistently overweights recent experiences (recency bias) will provide teaching signals that distort the learned reward function. More fundamentally, CIRL assumes the human *wants* to teach the robot, but in practice humans may be inattentive, adversarial, or simply unaware that their behavior is being interpreted as preference data. The choice architecture effects noted above compound this issue: the system's interface can systematically bias the human's responses in ways that CIRL does not account for.

Looking Ahead: From Individual Decisions to Collective Preferences

This chapter developed methods for sequential decision-making with preference feedback from a **single** user. But what happens when **multiple** users have different preferences? Chapter 5 (Aggregation) addresses this challenge using social choice theory and mechanism design. The winner concepts introduced here—Condorcet, Borda, von Neumann—will reappear in that context. In particular, a key result in Chapter 5 shows that DPO implicitly optimizes the **Borda score** when aggregating across annotators, connecting the dueling bandit framework developed here to the social choice theory of preference aggregation. Chapter 6 then asks the normative question: *whose* preferences should we optimize for, and what fairness constraints should apply?

4.9 Summary

- This chapter shifts perspective from **measuring** preferences (Chapter 3) to **making decisions** under preference uncertainty—managing uncertainty to maximize cumulative reward rather than merely reducing it.
- **Thompson Sampling** ("sample a plausible world, act optimally within it") naturally balances exploration and exploitation by sampling from the posterior. Combined with the Laplace approximation, it handles both linear and nonlinear (GP-based) preference models.
- **Dueling bandits** formalize decision-making when only pairwise comparisons are observable. Three winner concepts—**Condorcet** (beats all others), **Borda** (highest average win rate), and **von Neumann** (minimax optimal mixed strategy)—can diverge, and the choice of winner concept shapes what "regret" means.
- **Preferential Bayesian Optimization (PBO)** tackles the continuous-armed case where the goal is to find the best option in a large or infinite space using only preference feedback. The **qEUBO** acquisition function provably converges while the seemingly natural **qEI** can fail.
- Regret bounds for PBO scale as $R_T = O(\sqrt{\beta_T \gamma_T T})$, where γ_T is the maximum information gain—providing convergence guarantees that depend on the kernel's complexity.
- **Cooperative Inverse Reinforcement Learning (CIRL)** models humans as strategic agents who actively teach rather than passively demonstrate, enabling more efficient preference communication through cooperative interaction.
- **(Optional)** The **policy gradient theorem** provides the mathematical bridge from bandit methods to full RL-based preference optimization. **GRPO** eliminates the need for a learned value function by using group-relative advantages, connecting RL optimization back to the pairwise comparison paradigm.
- As system designers, we are **choice architects**: order effects, framing, and defaults influence human choices in ways that standard models (IIA) do not capture, requiring awareness of these biases in deployment.

4.10 Quick Check

Test your understanding before proceeding to the exercises.

1. Thompson Sampling draws from the posterior and acts greedily under the sample. Why does this produce exploration, even though each individual decision is "greedy"?
2. Construct a 3-arm preference matrix where the Condorcet winner and Borda winner differ. (Hint: the Condorcet winner beats all others head-to-head but may have low average win rates.)
3. Why does the qEI acquisition function fail to converge in some settings while qEUBO provably converges? What is the key difference in their objectives?
4. In CIRL, what changes if the human acts as a *teacher* (choosing informative demonstrations) rather than a *demonstrator* (acting optimally for themselves)?

5. (Optional) The policy gradient theorem uses $\nabla_\theta \log \pi_\theta(a \mid s)$ to estimate the gradient of the RL objective. Why do the environment's transition dynamics $P(s' \mid s, a)$ not appear in the gradient, even though they affect the trajectory distribution?

Essential Reading

If you read only five papers from this chapter's references, make them these:

1. **Thompson (1933)** — Thompson Sampling: the original 1933 paper on probability matching for sequential decisions
2. **Russo et al. (2018)** — A modern tutorial on Thompson Sampling with theory and applications
3. **Sui et al. (2017)** — Multi-dueling bandits: extending pairwise comparisons to multi-way preference feedback
4. **Brochu, Cora, and Freitas (2010)** — A tutorial on Bayesian optimization, the foundation for preferential BO
5. **Hadfield-Menell et al. (2016)** — CIRL: cooperative inverse reinforcement learning for value alignment
6. **Sutton and Barto (2018)** — Reinforcement Learning: An Introduction, the standard reference for MDP formulations and policy gradient methods
7. **Shao et al. (2024)** — DeepSeekMath: the paper introducing GRPO for LLM alignment

4.11 Discussion Questions

1. **Measurement vs. Optimization**: How does the reward maximization perspective differ from the measurement perspective on preference learning? When might one approach be more appropriate than the other?

2. **Thompson Sampling Intuition**: Explain in your own words how Thompson Sampling achieves the exploration-exploitation tradeoff. Why does sampling from the posterior distribution lead to exploration in uncertain regions?

3. **Laplace Approximation**: Why is the Laplace approximation necessary for Thompson Sampling with binary feedback? What are the potential limitations of this approximation?

4. **Regret Definitions**: Compare and contrast strong regret, weak regret, and Bayesian regret in the dueling bandit setting. Give an example scenario where each would be the most appropriate metric.

5. **Winner Concepts**: When might the Condorcet winner, Borda winner, and Von Neumann winner diverge? Construct a simple example with 3-4 alternatives where these concepts give different answers.

6. **Acquisition Functions**: What are the key tradeoffs between the UCB and Thompson Sampling acquisition functions? How does Pure Exploration differ in its objectives?

7. **qEUBO vs. qEI**: Why does qEUBO provably converge while qEI can fail to converge under the same assumptions? What does this suggest about designing acquisition functions for preferential feedback?
8. **Copeland Score**: How does the soft-Copeland score relate to finding the Condorcet winner? Why might we prefer the soft version over the hard indicator-based version?
9. **Contextual Bandits**: How do contextual bandits extend the basic multi-armed bandit framework? Give three real-world examples where context is crucial for making good decisions.
10. **Exploration in Practice**: In recommendation systems, how might information asymmetry be used ethically to encourage exploration? What are the potential risks of this approach?
11. **Bayesian Optimization Connection**: Explain the connection between Preferential Bayesian Optimization and traditional Bayesian optimization. How does the preference feedback model change the inference problem?
12. **Scalability Challenges**: What are the main computational challenges in scaling dueling bandit and PBO algorithms to large action spaces? How might these be addressed?
13. **(Optional) GRPO vs. DPO**: Both GRPO and DPO optimize a policy using preference data. DPO is "offline" (optimizes directly on a fixed dataset), while GRPO is "online" (generates new samples during training). What are the tradeoffs? When might each be preferred?
14. **(Optional) Variance Reduction**: GRPO uses a group-relative baseline while classical REINFORCE uses a learned value function baseline. Compare the computational and statistical tradeoffs. How does the group size G affect the bias-variance tradeoff?

4.12 Further Reading

For readers who want to go deeper into the topics introduced in this chapter, we recommend the following:

- **Russo et al. (2018)**, "A Tutorial on Thompson Sampling" — A modern tutorial covering the theory, applications, and extensions of Thompson Sampling, including connections to information-directed sampling and contextual bandits.
- **Bengs et al. (2021)**, "Preference-Based Online Learning with Dueling Bandits: A Survey" — A comprehensive survey of dueling bandit algorithms and their regret bounds, covering Condorcet, Borda, and von Neumann winner concepts and the algorithms designed for each.
- **Lattimore and Szepesvári (2020)**, *Bandit Algorithms* — The definitive modern textbook on bandit theory, providing rigorous treatments of regret bounds, exploration strategies, and the information-theoretic limits of sequential decision-making.

- **Shahriari et al. (2016)**, "Taking the Human Out of the Loop: A Review of Bayesian Optimization" — A survey connecting Bayesian optimization to preference learning, covering acquisition functions, GP surrogate models, and applications to hyperparameter tuning and experimental design.
- **Hadfield-Menell et al. (2016)**, "Cooperative Inverse Reinforcement Learning" — The foundational CIRL paper, formalizing the cooperative game between a human and a robot that learns the human's reward function through interaction.
- **Sutton and Barto (2018)**, *Reinforcement Learning: An Introduction* — The standard RL textbook, essential for readers who want to understand the MDP foundations and policy gradient methods discussed in the optional RL section of this chapter.
- **Thaler and Sunstein (2008)**, *Nudge: Improving Decisions About Health, Wealth, and Happiness* — The foundational work on choice architecture, documenting how presentation order, framing, and defaults influence human choices in ways that standard preference models do not capture.
- **Kahneman and Tversky (1979)**, "Prospect Theory: An Analysis of Decision Under Risk" — The seminal paper on how humans systematically deviate from expected utility maximization, providing the behavioral foundations for why pairwise comparisons (PBO) are often more reliable than absolute ratings.

4.13 Exercises

4.13.1 Basic (*)

1. **Posterior Sampling**: Given a logistic regression model with prior $U \sim \mathcal{N}(0, I)$ and two observations $(V_1, Y_1 = 1)$ and $(V_2, Y_2 = 0)$ with $V_1 = [1, 0]^\top$ and $V_2 = [0, 1]^\top$, write down the log-posterior and identify the form of the Laplace approximation.

2. **Regret Calculation**: Consider a dueling bandit with 3 arms where the preference matrix is:

$$P = \begin{pmatrix} 0.5 & 0.6 & 0.7 \\ 0.4 & 0.5 & 0.6 \\ 0.3 & 0.4 & 0.5 \end{pmatrix}$$

   Identify the Condorcet winner and compute the expected regret of always choosing arms $(2, 3)$.

3. **Copeland Score**: For the preference matrix in Exercise 2, compute the Copeland score for each arm.

4. **(Optional) MDP for LLM Generation**: Formally define the MDP for autoregressive text generation. What is the state space? Action space? When is reward received? What is the discount factor? How does the horizon T relate to the generated sequence length?

4.13.2 Intermediate (**)

4. **GP Classification**: Implement GP classification with Laplace approximation for a 1D toy problem. Generate binary labels from a latent sinusoidal function passed through a sigmoid, then fit a GP and visualize the posterior mean and uncertainty.

5. **Thompson Sampling Simulation**: Implement Thompson Sampling for a linear bandit with Gaussian prior. Run simulations comparing Thompson Sampling to UCB and ε-greedy on a synthetic problem. Plot cumulative regret curves.

6. **Dueling Bandit Simulation**: Implement the Interleaved Filter algorithm and test it on a dueling bandit problem with 5 arms. Compare its sample complexity to uniform exploration for finding the Condorcet winner.

7. **(Optional) Policy Gradient Derivation**: Starting from the trajectory probability $p(\tau \mid \theta) = \rho_0(s_0) \prod_t \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$, derive the policy gradient theorem. Show that subtracting a state-dependent baseline $b(s_t)$ from the return does not bias the gradient estimate.

4.13.3 Advanced (***)

7. **Laplace Approximation Derivation**: Starting from the GP binary classification model, derive the full Laplace approximation including the Hessian computation for the posterior covariance. Show that the resulting approximation is Gaussian with the correct mean and covariance.

8. **qEI Failure Case**: Reproduce the proof that qEI can fail to converge by implementing the counterexample from the chapter. Verify empirically that qEI gets stuck while qEUBO converges.

9. **POP-BO Regret Bound**: Following the proof sketch in the chapter, derive the regret bound for POP-BO with a linear kernel. Explain how the covering number $\mathcal{N}(\mathcal{B}_f, \epsilon, \|\cdot\|_\infty)$ appears in the bound.

10. **Custom Acquisition Function**: Design and implement a novel acquisition function for preferential Bayesian optimization that balances exploration and exploitation in a different way than qEUBO. Provide theoretical justification or empirical evidence for its effectiveness.

11. **(Optional) GRPO Implementation**: Implement GRPO for a contextual bandit with Bradley-Terry rewards. Compare convergence and gradient variance against REINFORCE with a learned baseline for group sizes $G \in \{2, 4, 8, 16\}$. Investigate: for what group size does the marginal benefit of additional samples diminish?

12. **(Optional) KL-Constrained Optimization**: The RLHF objective includes a KL penalty $\beta\,\mathrm{KL}[\pi_\theta \| \pi_{\text{ref}}]$. Derive the optimal policy in closed form (it has the form $\pi^*(y \mid x) \propto \pi_{\text{ref}}(y \mid x) \exp(r(x, y)/\beta)$). Show that this is exactly the DPO implicit reward from Chapter 1.

References

Astudillo, Raul, Zi Wang Lin, Eytan Bakshy, and Peter Frazier. 2023. "qEUBO: A Decision-Theoretic Acquisition Function for Preferential Bayesian Optimization." In *International Conference on Artificial Intelligence and Statistics*, 1093–1114. PMLR.

Bastani, Hamsa, and Mohsen Bayati. 2020. "Online Decision Making with High-Dimensional Covariates." *Operations Research* 68 (1): 276–94. https://doi.org/10.1287/opre.2019.1902.

Bengs, Viktor, Róbert Busa-Fekete, Adil El Mesaoudi-Paul, and Eyke Hüllermeier. 2021. "Preference-Based Online Learning with Dueling Bandits: A Survey." *Journal of Machine Learning Research* 22 (7): 1–108.

Bouneffouf, Djallel, Amel Bouzeghoub, and Alda Lopes Gançarski. 2012. "A Contextual-Bandit Algorithm for Mobile Context-Aware Recommender System." In *Neural Information Processing*, edited by Tingwen Huang, Zhigang Zeng, Chuandong Li, and Chi Sing Leung, 324–31. Berlin, Heidelberg: Springer Berlin Heidelberg.

Bouneffouf, Djallel, Irina Rish, and Charu Aggarwal. 2020. "Survey on Applications of Multi-Armed and Contextual Bandits." In *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. Glasgow, United Kingdom: IEEE Press. https://doi.org/10.1109/CEC48606.2020.9185782.

Brochu, Eric, Vlad M. Cora, and Nando de Freitas. 2010. "A Tutorial on Bayesian Optimization of Expensive Cost Functions." *arXiv Preprint arXiv:1012.2599*.

Hadfield-Menell, Dylan, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. "Cooperative Inverse Reinforcement Learning." *Advances in Neural Information Processing Systems* 29.

Kahneman, Daniel, and Amos Tversky. 1979. "Prospect Theory: Analysis of Decision Under Risk." *Econometrica* 47 (2). https://doi.org/10.2307/1914185.

Lattimore, Tor, and Csaba Szepesvári. 2020. *Bandit Algorithms*. Cambridge: Cambridge University Press.

Liu, Bing, Tong Yu, Ian Lane, and Ole J. Mengshoel. 2018. "Customized Nonlinear Bandits for Online Response Selection in Neural Conversation Models." In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press.

Russo, Daniel J., Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. 2018. "A Tutorial on Thompson Sampling." *Foundations and Trends in Machine Learning* 11 (1): 1–96.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. "Trust Region Policy Optimization." In *International Conference on Machine Learning*, 1889–97.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. "Proximal Policy Optimization Algorithms." *arXiv Preprint arXiv:1707.06347*.

Shahriari, Bobak, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. "Taking the Human Out of the Loop: A Review of Bayesian Optimization." *Proceedings of the IEEE* 104 (1): 148–75.

Shao, Zhihong, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." *arXiv Preprint arXiv:2402.03300*.

Sui, Yanan, Vincent Zhuang, Joel W. Burdick, and Yisong Yue. 2017. "Multi-Dueling Bandits with Dependent Arms." In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Sui, Yanan, Masrour Zoghi, Katja Hofmann, and Yisong Yue. 2018. "Advancements in Dueling Bandits." *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. https://doi.org/10.24963/ijcai.2018/776.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book.

Thaler, Richard H., and Cass R. Sunstein. 2008. *Nudge: Improving Decisions about Health, Wealth, and Happiness*. New Haven, CT: Yale University Press.

Thompson, William R. 1933. "On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples." *Biometrika* 25 (3–4): 285–94.

Williams, Ronald J. 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement

Learning." *Machine Learning* 8 (3): 229–56.
Wu, Jian, and Peter I. Frazier. 2018. "The Parallel Knowledge Gradient Method for Batch Bayesian Optimization." https://arxiv.org/abs/1606.04414.
Xu, Wenjie, Wenbin Wang, Yuning Jiang, Bratislav Svetozarevic, and Colin N. Jones. 2024. "Principled Preferential Bayesian Optimization." https://arxiv.org/abs/2402.05367.
Yue, Yisong, Josef Broder, Robert Kleinberg, and Thorsten Joachims. 2012. "The k-Armed Dueling Bandits Problem." *Journal of Computer and System Sciences* 78 (5): 1538–56. https://doi.org/https://doi.org/10.1016/j.jcss.2011.12.028.
Yue, Yisong, and Thorsten Joachims. 2009. "Interactively Optimizing Information Retrieval Systems as a Dueling Bandits Problem." *Proceedings of the 26th Annual International Conference on Machine Learning*. https://doi.org/10.1145/1553374.1553527.
Zhang, Tong. 2021. "Feel-Good Thompson Sampling for Contextual Bandits and Reinforcement Learning." *CoRR* abs/2110.00871. https://arxiv.org/abs/2110.00871.
Zhou, Qian, XiaoFang Zhang, Jin Xu, and Bin Liang. 2017. "Large-Scale Bandit Approaches for Recommender Systems." In *Neural Information Processing*, edited by Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, 811–21. Cham: Springer International Publishing.
Zoghi, Masrour, Zohar S. Karnin, Shimon Whiteson, and Maarten De Rijke. 2015. "Copeland Dueling Bandits." In *Advances in Neural Information Processing Systems*. Vol. 28.

5 Aggregation

Intended Learning Outcomes

By the end of this chapter you will be able to:

- **Distinguish** between social welfare functions and social choice functions, and explain their role in aggregating preferences from multiple agents.
- **State** Arrow's Impossibility Theorem and the Gibbard–Satterthwaite Theorem, explaining why no "perfect" voting rule exists for three or more alternatives.
- **Define** single-peaked preferences and derive the generalized median voter scheme as a strategy-proof aggregation rule on restricted domains.
- **Apply** Moulin's theorem to design strategy-proof, Pareto efficient voting rules when preferences lie on a single-peaked domain.
- **Connect** the Borda count to Direct Preference Optimization (DPO), showing that the DPO-optimal policy maximizes the expected Borda score.
- **Identify** nosy vs. non-nosy preferences and apply Sen's framework to determine when liberal vs. illiberal assistance is appropriate in AI systems.
- **Analyze** the tension between privacy and personalization in preference learning using the Contextual Integrity framework.
- **Recognize** the inversion problem (observed behavior does not necessarily equal underlying preferences) and its implications for revealed preference approaches.
- **Design** incentive-compatible mechanisms for settings without monetary transfers, including peer prediction and online learning.
- **Evaluate** real-world aggregation systems like Community Notes for combining diverse perspectives on content moderation.

Suggested Lecture Plan

This chapter can be covered in four 50-minute lectures plus a discussion section:

Lecture 1 (Sections 5.1–5.2): Classical Social Choice Theory

- Motivation: When and why must we aggregate preferences? (10 min)
- Voting rules: Plurality, Borda, STV, Condorcet (10 min)
- Arrow's theorem and Gibbard–Satterthwaite (20 min)
- Implications for AI alignment and RLHF (10 min)

Lecture 2 (Sections 5.3–5.4): Escaping Impossibility

- Domain restrictions: Single-peaked preferences (15 min)
- Generalized median voter and Moulin's theorem (15 min)

– Borda count and modified IIA (IIA') (10 min)
– Code demo: DPO-Borda connection (10 min)

Lecture 3 (Sections 5.5–5.7): Beyond Classical Voting

– Multi-issue voting and separable preferences (10 min)
– Nosy preferences and the Paretian Liberal paradox (15 min)
– Case study: Community Notes model (15 min)
– Discussion: When should systems override stated preferences? (10 min)

Lecture 4 (Sections 5.8–5.10): Challenges in Practice

– The inversion problem: Behavior vs. mental states (15 min)
– Privacy and Contextual Integrity (15 min)
– Mechanism design: Auctions and VCG (20 min)

Discussion Section: Peer prediction, incentive-compatible online learning, and exercises

Chapters 1–4 focused on modeling preferences of a *single* decision-maker or learning from the feedback of individual users. But many real-world applications require aggregating preferences across *multiple* individuals. When training a language model with RLHF, annotators may disagree about which response is better. When a recommender system serves millions of users, it must balance diverse tastes. When an AI assistant helps a family plan a vacation, it must somehow combine different members' preferences.

This chapter explores the fundamental challenges and solutions for preference aggregation. We begin with classical social choice theory, which reveals deep impossibility results: there is no "perfect" way to aggregate preferences. We then examine how to escape these impossibilities through domain restrictions, scoring rules, and mechanism design. Along the way, we connect these classical results to modern AI alignment challenges, including the relationship between the Borda count and DPO, the problem of nosy preferences in content moderation, and the tension between privacy and personalization.

5.1 Chapter Overview

This chapter is organized into five parts:

1. **Social Choice Theory** (): We introduce the formal framework of social choice, present Arrow's and Gibbard–Satterthwaite's impossibility theorems, and discuss their implications for AI systems that aggregate human feedback.
2. **Escaping Impossibility** (): We show how restricting the domain of preferences (e.g., to single-peaked preferences) enables strategy-proof aggregation, and we prove a key connection between the Borda count and DPO.
3. **Beyond Classical Voting** (–): We extend to multi-issue settings, discuss nosy preferences and the Paretian Liberal paradox, and analyze Community Notes as a real-world case study.

4. **Challenges in Practice** (–): We examine the inversion problem (behavior □ preferences), privacy concerns in preference learning, and when paternalistic intervention may be justified.
5. **Mechanism Design** (–): We cover auctions, VCG mechanisms, optimal auction design, peer prediction, and incentive-compatible online learning.

5.2 Social Choice Theory

In many applications, human preferences must be aggregated across multiple individuals to determine a collective decision or ranking. This process is central to social choice theory, which provides a mathematical foundation for preference aggregation. Unlike individual preference modeling, which focuses on how a single person makes decisions, social choice theory addresses the challenge of combining multiple preference profiles into a single coherent outcome. A social welfare function (SWF) takes as input each individual's preference ranking over a set of alternatives and produces a social ranking of those alternatives. A related concept is a social choice function (SCF), which selects a single winning alternative given individuals' preferences. Many voting rules can be seen as social choice functions that aim to reflect the group's preferences. Formally, let $N = \{1, 2, \dots, n\}$ be a set of n voters (agents) and $A = \{a_1, \dots, a_m\}$ a set of m alternatives (with $m \geq 3$). Each voter i has a preference order $\succ_i$ over A. A social choice function is a mapping $f : (\succ_1, \dots, \succ_n) \mapsto A$ that picks a winning alternative for each possible profile of individual preferences. A social welfare function is a mapping that produces a complete societal ranking $\succ^*$ of the alternatives. The central question is: can we design an aggregation rule that faithfully represents individual preferences while satisfying certain fairness or rationality axioms?

Many common voting rules illustrate different methods of aggregation, each with its own merits and vulnerabilities:

- Plurality: Each voter names their top choice; the alternative with the most votes wins.
- Borda Count: Voters rank all alternatives, and points are assigned based on the position in each ranking. For example, with m alternatives, a voter's top-ranked alternative gets $m - 1$ points, the second-ranked gets $m - 2$, and so on down to 0. The Borda score of an alternative is the sum of points from all voters, and the winner is the alternative with the highest total score.
- Single Transferable Vote (STV): Voters rank choices, and the count proceeds in rounds. In each round, the alternative with the fewest votes is eliminated and those votes are transferred to the next preferred remaining alternative on each ballot, until one candidate has a majority.
- Condorcet Methods: These look for a candidate that wins in all pairwise majority contests against other alternatives (the Condorcet winner), if such an alternative exists.

However, preference aggregation is not always straightforward. The Condorcet paradox illustrates that majority preferences can be cyclic (rock-paper-scissors style), so that no single alternative is majority-preferred to all others, violating transitivity. Different voting rules can

yield different winners on the same profile, highlighting how the choice of rule influences the outcome.

The following code demonstrates how the same set of voter preferences can produce different winners under different voting rules—a phenomenon that underscores why the choice of aggregation method is itself a value judgment.

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

def plurality_winner(rankings, alternatives):
    """Plurality: each voter's top choice gets one vote."""
    first_place_counts = {a: 0 for a in alternatives}
    for ranking in rankings:
        first_place_counts[ranking[0]] += 1
    return max(first_place_counts, key=first_place_counts.get), first_place_counts

def borda_winner(rankings, alternatives):
    """Borda count: m-1 points for 1st, m-2 for 2nd, etc."""
    m = len(alternatives)
    scores = {a: 0 for a in alternatives}
    for ranking in rankings:
        for pos, alt in enumerate(ranking):
            scores[alt] += (m - 1 - pos)
    return max(scores, key=scores.get), scores

def condorcet_winner(rankings, alternatives):
    """Find Condorcet winner (beats all others in pairwise majority), if one
    exists."""
    m = len(alternatives)
    pairwise = {a: {b: 0 for b in alternatives} for a in alternatives}
    for ranking in rankings:
        for i, a in enumerate(ranking):
            for b in ranking[i+1:]:
                pairwise[a][b] += 1
    n = len(rankings)
    for a in alternatives:
        if all(pairwise[a][b] > n / 2 for b in alternatives if b != a):
            return a, pairwise
    return None, pairwise

# Example: 7 voters, 3 alternatives where different rules disagree
alternatives = ['A', 'B', 'C']
rankings = [
    ['A', 'B', 'C'],  # Voter 1
    ['A', 'B', 'C'],  # Voter 2
    ['A', 'B', 'C'],  # Voter 3
    ['B', 'C', 'A'],  # Voter 4
```

```
    ['B', 'C', 'A'],  # Voter 5
    ['C', 'B', 'A'],  # Voter 6
    ['C', 'B', 'A'],  # Voter 7
]

plur_win, plur_scores = plurality_winner(rankings, alternatives)
borda_win, borda_scores = borda_winner(rankings, alternatives)
condorcet_win, pairwise = condorcet_winner(rankings, alternatives)

print("=== 7 Voters, 3 Alternatives ===")
print(f"Rankings: 3×(A>B>C), 2×(B>C>A), 2×(C>B>A)\n")
print(f"Plurality winner: {plur_win}  (votes: {plur_scores})")
print(f"Borda winner:     {borda_win}  (scores: {borda_scores})")
print(f"Condorcet winner: {condorcet_win or 'None (cycle!)'}")
print(f"\nPairwise margins:")
for a in alternatives:
    for b in alternatives:
        if a != b:
            print(f"  {a} vs {b}: {pairwise[a][b]}-{pairwise[b][a]}",
                  " " if pairwise[a][b] > len(rankings)/2 else "")

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(8, 3.5))

# Left: Plurality vs Borda scores
x = np.arange(len(alternatives))
width = 0.35
ax = axes[0]
bars1 = ax.bar(x - width/2, [plur_scores[a] for a in alternatives], width,
               label='Plurality', color='steelblue', alpha=0.8)
bars2 = ax.bar(x + width/2, [borda_scores[a] for a in alternatives], width,
               label='Borda', color='coral', alpha=0.8)
ax.set_xticks(x)
ax.set_xticklabels(alternatives, fontsize=12)
ax.set_ylabel('Score')
ax.set_title('Same voters, different winners')
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

# Right: Condorcet paradox - 3 voters creating a cycle
cycle_rankings = [
    ['A', 'B', 'C'],  # Voter 1: A > B > C
    ['B', 'C', 'A'],  # Voter 2: B > C > A
    ['C', 'A', 'B'],  # Voter 3: C > A > B
]
_, cycle_pairwise = condorcet_winner(cycle_rankings, alternatives)

ax = axes[1]
theta = np.linspace(0, 2*np.pi, 4)[:-1] + np.pi/2
```

```
positions = {a: (np.cos(t), np.sin(t)) for a, t in zip(alternatives, theta)}
for a in alternatives:
    ax.scatter(*positions[a], s=400, zorder=5, color='steelblue')
    ax.annotate(a, positions[a], fontsize=14, fontweight='bold', ha='center',
    ↪  va='center', color='white')

# Draw arrows for majority preferences
arrows = [('A', 'B'), ('B', 'C'), ('C', 'A')]  # The cycle
for a, b in arrows:
    ax.annotate('', xy=positions[b], xytext=positions[a],
                arrowprops=dict(arrowstyle='->', color='red', lw=2,
                                connectionstyle='arc3,rad=0.15'))
    mid = ((positions[a][0]+positions[b][0])/2, (positions[a][1]+positions[b][1])/2)
    ax.annotate('2-1', mid, fontsize=9, ha='center', va='center',
                bbox=dict(boxstyle='round,pad=0.2', facecolor='lightyellow'))

ax.set_title('Condorcet paradox: A→B→C→A')
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-1.5, 1.5)
ax.set_aspect('equal')
ax.axis('off')

plt.tight_layout()
plt.show()

print("\nCondorcet paradox: With 3 voters (A>B>C, B>C>A, C>A>B),")
print("majority rule cycles: A beats B (2-1), B beats C (2-1), C beats A (2-1).")
print("No Condorcet winner exists - every alternative loses to some other.")
``` To guide the design of social choice functions, several desirable properties or
↪  axioms have been proposed. Three classical fairness criteria are:

1. Unanimity (Pareto efficiency): If all individuals strictly prefer one alternative
↪  $x$ over another $y$ (i.e. $x \succ_i y$ for every voter $i$), then the group
↪  ranking should prefer $x$ over $y$ as well ($x \succ^* y$).
2. Independence of Irrelevant Alternatives (IIA): The social preference between any
↪  two alternatives $x$ and $y$ should depend only on the individual preferences
↪  between $x$ and $y$. In other words, if we change individuals' rankings of other
↪  "irrelevant" alternatives (not $x$ or $y$) in any way, the group's relative
↪  ordering of $x$ and $y$ should remain unchanged.
3. Non-dictatorship: The aggregation should not simply follow a single individual's
↪  preference regardless of others. There is no voter $i$ who always gets their top
↪  choice as the social choice (or whose rankings always become the social ranking),
↪  irrespective of other voters' preferences.
```
```

```
Additionally, we assume an unrestricted domain (universal admissibility): individuals may have any transitive preference ordering over the $m$ alternatives (no restrictions like single-peaked preferences unless explicitly imposed). One might hope that a fair voting rule exists that satisfies all the above properties for three or more alternatives. Surprisingly, a seminal negative result shows this is impossible.

::: {.callout-caution title="Worked Example: Same Voters, Three Different Winners" collapse="false"}
Consider 7 RLHF annotators ranking three model responses $\{A, B, C\}$ to a prompt:

| Annotators | Ranking |
|:---:|:---:|
| 3 annotators | $A \succ B \succ C$ |
| 2 annotators | $B \succ C \succ A$ |
| 2 annotators | $C \succ B \succ A$ |

**Plurality:** Count first-place votes: $A$ gets 3, $B$ gets 2, $C$ gets 2. Winner: **A**.

**Borda Count:** With 3 alternatives, assign 2 points for 1st, 1 for 2nd, 0 for 3rd:

- $A$: $3 \times 2 + 2 \times 0 + 2 \times 0 = 6$
- $B$: $3 \times 1 + 2 \times 2 + 2 \times 1 = 9$
- $C$: $3 \times 0 + 2 \times 1 + 2 \times 2 = 6$

Winner: **B** (with 9 points).

**Condorcet:** Check pairwise majorities:

- $A$ vs $B$: 3 prefer $A$, 4 prefer $B$ $\Rightarrow$ $B$ wins
- $A$ vs $C$: 3 prefer $A$, 4 prefer $C$ $\Rightarrow$ $C$ wins
- $B$ vs $C$: 5 prefer $B$, 2 prefer $C$ $\Rightarrow$ $B$ wins

$B$ beats both $A$ and $C$ in pairwise majority. Condorcet winner: **B**.

**Takeaway:** Plurality selects $A$ (most first-place votes), but $A$ actually *loses* to both $B$ and $C$ in head-to-head matchups! The Borda and Condorcet methods agree on $B$, which has broader support. In RLHF, this matters: if annotator preferences are aggregated via majority-of-first-choices (like "pick the best"), the selected response may not be the one with the broadest appeal.
:::

#### Arrow's Impossibility Theorem {#sec-arrow}
```

```
158 Arrow's Impossibility Theorem [@arrow1951] is a cornerstone of social choice theory.
    ↪ It states that when there are three or more alternatives ($m\ge 3$), no social
    ↪ welfare function can simultaneously satisfy Unanimity, IIA, and Non-dictatorship -
    ↪ unless it is a trivial dictatorial rule. In other words, any aggregation mechanism
    ↪ that is not dictatorial will inevitably violate at least one of the fairness
    ↪ criteria. The theorem is usually proven by contradiction: assuming a social
    ↪ welfare function satisfies all conditions, one can show that one voter's
    ↪ preferences always decide the outcome, hence the rule is dictatorial. Intuitively,
    ↪ Arrow's theorem is driven by the possibility of preference cycles in majority
    ↪ voting. Even if individual preferences are transitive, aggregated majorities can
    ↪ prefer $A$ to $B$, $B$ to $C$, and $C$ to $A$ in a cycle, as in the Condorcet
    ↪ paradox. Under Unanimity and IIA, the social ranking must locally match these
    ↪ pairwise preferences, but this produces a contradiction with transitivity unless
    ↪ one voter's ranking is given overriding authority. A sketch of Arrow's proof is as
    ↪ follows: one shows that under the axioms, the social ranking between any two
    ↪ alternatives $x$ and $y$ must agree with some particular voter's preference (the
    ↪ "pivotal" voter for that pair). With IIA, the identity of the pivotal voter must
    ↪ be the same across all pairs of alternatives, otherwise by cleverly constructing
    ↪ profiles one can derive a conflict. This single pivotal voter then effectively
    ↪ dictates the entire social order, violating Non-dictatorship. Hence, the axioms
    ↪ are incompatible.
159
160 ::: {.callout-warning title="Common Pitfall: Overgeneralizing Arrow's Theorem"}
161 Arrow's theorem applies specifically to **social welfare functions** that produce a
    ↪ complete ranking from ordinal preferences over three or more alternatives. A
    ↪ common misconception is that "all preference aggregation is impossible." In fact,
    ↪ Arrow's theorem does **not** directly apply to (1) cardinal scores (e.g., 1-5
    ↪ ratings), (2) pairwise comparison models that output partial orders rather than
    ↪ complete rankings, or (3) domains with restricted preferences (e.g., single-peaked
    ↪ preferences, where majority rule is well-behaved). In RLHF, we typically work with
    ↪ cardinal reward models or pairwise Bradley-Terry likelihoods, which sidestep
    ↪ Arrow's specific impossibility - but other impossibilities (Gibbard-Satterthwaite)
    ↪ still constrain what's achievable. The theorem tells us we must **sacrifice at
    ↪ least one axiom**, not that aggregation is hopeless.
162 :::
163
164 ::: {.callout-important title="Ordinal vs. Cardinal: What Escapes Arrow's Theorem"}
165 Arrow's theorem applies to **ordinal** aggregation: it constrains rules that take
    ↪ ranked preferences as input. However, when we have **cardinal** utilities---as in
    ↪ Bradley-Terry models, where each item has a numerical score $V_j$---utilitarian
    ↪ aggregation (summing or averaging utilities) is well-defined and not subject to
    ↪ Arrow's impossibility. The catch is **interpersonal comparability**: summing
    ↪ utilities across people assumes that "$V_j = 5$ for Alice" means the same thing as
    ↪ "$V_j = 5$ for Bob," which is a strong and often unjustified assumption. In RLHF,
    ↪ treating all annotators' Bradley-Terry parameters as comparable implicitly makes
    ↪ this assumption. When annotators have different scales or noise levels, naive
    ↪ averaging can systematically favor annotators whose scores have larger variance.
166 :::
```

```
167
168 Arrow's Impossibility Theorem has profound implications: it formalizes the inherent
     ↪ trade-offs in designing any fair aggregation scheme. In practice, different voting
     ↪ rules relax one or more of Arrow's conditions. For instance, Borda count violates
     ↪ IIA (since introducing or removing an irrelevant alternative can change the point
     ↪ totals), while a dictatorship violates fairness blatantly. The theorem suggests
     ↪ that every practical voting system must sacrifice at least one of the ideal
     ↪ fairness criteria. It also motivated the exploration of alternative frameworks
     ↪ (such as allowing interpersonal comparisons of utility or cardinal preference
     ↪ aggregation) to escape the impossibility by weakening assumptions.
169
170 Complementing Arrow's theorem, the Gibbard-Satterthwaite theorem focuses on incentives
     ↪ and strategic manipulation in voting systems [@gibbard1973; @satterthwaite1975].
     ↪ It considers any deterministic social choice function $f$ that chooses a single
     ↪ winner from the set of $m\ge 3$ alternatives. The theorem states that if $f$ is
     ↪ strategy-proof (incentive compatible) and onto (its range of outcomes is the
     ↪ entire set of alternatives), then $f$ must be dictatorial. Strategy-proofness
     ↪ (also called truthfulness or dominant-strategy incentive compatibility) means that
     ↪ no voter can ever benefit by misrepresenting their true preferences, regardless of
     ↪ what others do. In other words, reporting their genuine ranking is a (weakly)
     ↪ dominant strategy for each voter. The theorem implies that for any realistic
     ↪ voting rule where every alternative can possibly win, either one voter effectively
     ↪ decides the outcome (a dictatorship) or else the rule is susceptible to strategic
     ↪ manipulation by voters. The Gibbard-Satterthwaite theorem tells us that every
     ↪ non-dictatorial voting rule for 3 or more alternatives is manipulable: there will
     ↪ exist some election scenario where a voter can gain a more preferred outcome by
     ↪ voting insincerely (i.e. not according to their true preferences). For example, in
     ↪ a simple plurality vote, a voter whose true favorite is a long-shot candidate
     ↪ might vote for a more viable candidate to avoid a worst-case outcome ("lesser of
     ↪ two evils" voting). In a Borda count election, voters might strategically raise a
     ↪ competitor in their ranking to push down an even stronger rival. The only way to
     ↪ avoid all such strategic voting incentives is to have a dictatorship or limit the
     ↪ choice set to at most two alternatives.
171
172 The proof of Gibbard-Satterthwaite is non-trivial, but one can outline the idea: Given
     ↪ a non-dictatorial and onto rule $f$, one shows there exist at least three distinct
     ↪ outcomes that can result from some preference profiles. By carefully constructing
     ↪ profiles and using the onto property, one finds a situation where a single voter
     ↪ can change the outcome by switching their order of two candidates, demonstrating
     ↪ manipulability. The theorem is robust - even if we allow ties or weaker
     ↪ conditions, similar impossibility results hold (Gibbard's 1978 extension handles
     ↪ randomized rules). The practical takeaway is that all meaningful voting protocols
     ↪ encourage tactical voting in some situations. Nonetheless, certain systems are
     ↪ considered "harder to manipulate" or more resistant due to complexity or
     ↪ uncertainty. For instance, while STV (ranked-choice voting) can be manipulated in
     ↪ theory, determining a beneficial strategic vote can be NP-hard in worst cases,
     ↪ which arguably provides some practical deterrence to manipulation
     ↪ [@bartholdi1989].
```

```
173
174 Arrow's and Gibbard-Satterthwaite's theorems highlight the limitations any preference
     ↪ aggregation method must face. In domains like reinforcement learning from human
     ↪ feedback (RLHF) and AI alignment, we also aggregate preferences - often
     ↪ preferences of multiple human evaluators or preferences revealed in pairwise
     ↪ comparisons - to guide machine learning systems. While these settings sometimes
     ↪ use cardinal scores or learned reward functions (escaping the strict ordinal
     ↪ framework of Arrow's theorem), the spirit of these impossibility results still
     ↪ applies: there is no perfect way to aggregate human opinions without trade-offs.
175
176 For example, aggregating human feedback to train a model may run into inconsistencies
     ↪ analogous to preference cycles, especially when feedback comes from diverse
     ↪ individuals with different values. A simple majority vote over preferences might
     ↪ yield unstable or unfair outcomes if some annotators are systematically in the
     ↪ minority. Weighting votes by some credibility or expertise (weighted voting) can
     ↪ improve outcomes but raises the question of how to set the weights without
     ↪ introducing dictator-like influence. Recent research has proposed methods like
     ↪ jury learning - which integrates dissenting voices by having a panel ("jury") of
     ↪ models or human subgroups whose aggregated judgment guides the learning
     ↪ [@gordon2022jury] - to ensure minority preferences are not entirely ignored.
     ↪ Another perspective is social choice in AI alignment, which suggests using social
     ↪ choice theory to design AI systems that respect a plurality of human values
     ↪ instead of collapsing everything into a single objective. In pluralistic value
     ↪ alignment, instead of forcing a single "best" solution, an AI might present a
     ↪ diverse set of options or behave in a way that reflects a distribution of values.
     ↪ This approach aims to preserve the diversity of human preferences rather than
     ↪ always aggregating to one monolithic preference. For instance, a conversational AI
     ↪ might be designed to recognize multiple acceptable responses (each aligning with
     ↪ different value systems) rather than one canonical "aligned" response for a given
     ↪ query.
177
178 These considerations are especially relevant in generative AI and large language
     ↪ models, where training involves human preference data. If we aggregate feedback
     ↪ naively, we might overfit to the majority preference and lose minority
     ↪ perspectives (a form of tyranny of the majority). On the other hand, trying to
     ↪ satisfy everyone can lead to indecision or an incoherent objective. The
     ↪ impossibility results remind us there is no free lunch: we must carefully decide
     ↪ which properties to prioritize (e.g. giving more weight to expert annotators
     ↪ versus preserving broader fairness, or balancing consistency vs inclusivity).
     ↪ Designing aggregation mechanisms for AI that reflect collective human values is an
     ↪ ongoing challenge. It often involves insights from traditional voting theory (to
     ↪ understand trade-offs and failure modes) combined with machine learning techniques
     ↪ (to model and learn from preference data). In summary, social choice theory
     ↪ provides cautionary guidance as we build systems that learn from human
     ↪ preferences: we need to be conscious of which fairness criteria we relax and be
     ↪ transparent about the compromises being made in any preference aggregation
     ↪ pipeline.
179
```

```
## Escaping Impossibility: Domain Restrictions {#sec-domain-restrictions}

Arrow's and Gibbard-Satterthwaite's theorems assume an *unrestricted domain*: any transitive preference ordering is admissible. But what if we restrict the types of preferences that can occur? In many real-world settings, preferences have natural structure that we can exploit.

### Single-Peaked Preferences {#sec-single-peaked}

One of the most important restricted domains is *single-peaked preferences*. The idea is that alternatives can be arranged along a line (e.g., a left-right political spectrum, a temperature setting, or a budget level), and each voter has an "ideal point" or "peak" on this line, with preferences decreasing monotonically as we move away from the peak in either direction.

::: {.callout-note title="Definition: Single-Peaked Preferences"}
Let $Y$ be a totally ordered set of alternatives. A preference ordering $\succ$ over $Y$ is **single-peaked** if there exists a peak $p(\succ) \in Y$ such that for all $y, y' \in Y$:

- If $y < y' \leq p(\succ)$, then $y' \succ y$
- If $p(\succ) \leq y' < y$, then $y' \succ y$

The set of all single-peaked preferences on $Y$ is denoted $\text{SP}(Y)$.
:::

Intuitively, a voter with peak $p$ always prefers alternatives closer to $p$ over alternatives farther from $p$. This rules out preferences like "I prefer the extremes to the middle"-which would create the kind of cycles that drive Arrow's impossibility.

::: {.callout-warning title="When Is Single-Peakedness Reasonable?"}
Single-peakedness is a powerful escape from impossibility, but it requires alternatives to be ordered along a **single dimension**. This is natural for some settings: budget levels, temperature, or dosage amounts. However, it fails when preferences are inherently **multi-dimensional**---for instance, in LLM alignment, annotators may evaluate responses along multiple axes (helpfulness, harmlessness, conciseness, formality) with different trade-offs, producing multi-peaked preferences. In political settings, the assumption of a one-dimensional left-right spectrum is a simplification that breaks down for cross-cutting issues. Before relying on single-peaked domain restrictions, practitioners should verify that the assumption is empirically supported, for example by checking whether the preference data can be embedded in one dimension with low distortion.
:::
```

```
**Example: Temperature preferences.** Consider three colleagues deciding on the office thermostat setting. The alternatives are temperatures from 65°F to 75°F. Alice prefers 68°F (her peak), with utility decreasing as temperature moves away from 68. Bob prefers 72°F. Carol prefers 70°F. All three have single-peaked preferences on the temperature spectrum.

### Generalized Median Voter Scheme {#sec-median-voter}

On single-peaked domains, a remarkable class of voting rules emerges that satisfies strategy-proofness, Pareto efficiency, and other desirable properties.

::: {.callout-note title="Definition: Generalized Median Voter Scheme"}
Fix *phantom votes* $a_1, \ldots, a_{n-1} \in \mathbb{R} \cup \{\pm\infty\}$. The **generalized median voter scheme** is defined as:
$$
f(\succ_1, \ldots, \succ_n) = \text{median}\big(p(\succ_1), \ldots, p(\succ_n), a_1, \ldots, a_{n-1}\big)
$$ {#eq-median-voter}
where $p(\succ_i)$ is the peak of voter $i$'s preferences.
:::

The phantom votes act as "anchor points" that can shift the median in various directions. With $n-1$ phantom votes, there are $2n-1$ total values, so the median is well-defined. Different choices of phantom votes yield different voting rules:

- **Pure median rule**: Set all phantoms to $\pm\infty$ appropriately so they don't affect the median. The outcome is simply the median of the voters' peaks.
- **Dictatorial rule**: Set all phantoms equal to voter $i$'s peak. The outcome is always voter $i$'s ideal point.
- **Status quo rule**: Set all phantoms to the status quo point $q$. The outcome can only move toward the median of voter peaks if enough voters prefer change.

::: {.callout-important title="Moulin's Characterization Theorem"}
**Theorem (Moulin, 1980)**: On the domain of single-peaked preferences $\text{SP}(Y)$, a social choice function $f$ satisfies:

1. **Strategy-proofness (SP)**: No voter can benefit by misreporting their peak
2. **Pareto efficiency (PE)**: The outcome is never unanimously dispreferred to another alternative
3. **Anonymity and peaks-only**: The outcome depends only on the set of reported peaks

if and only if $f$ is a generalized median voter scheme.
:::

This is a powerful positive result: by restricting to single-peaked preferences, we escape Arrow's impossibility and can achieve both strategy-proofness and efficiency.
```

The following code demonstrates the generalized median voter scheme and its strategy-proofness:

```{pyodide-python}
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

def generalized_median(peaks, phantoms):
    """Compute generalized median voter outcome."""
    all_points = np.concatenate([peaks, phantoms])
    return np.median(all_points)

# Example: 5 voters with different peaks
voter_peaks = np.array([2.0, 4.0, 5.0, 7.0, 9.0])
phantoms = np.array([0.0, 3.0, 6.0, 10.0])  # 4 phantom votes

outcome = generalized_median(voter_peaks, phantoms)
print(f"Voter peaks: {voter_peaks}")
print(f"Phantom votes: {phantoms}")
print(f"Median outcome: {outcome}")

# Demonstrate strategy-proofness: voter 1 tries to manipulate
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Left plot: True reporting
ax = axes[0]
true_outcome = generalized_median(voter_peaks, phantoms)
ax.scatter(voter_peaks, np.zeros(5), s=100, c='blue', label='Voter peaks', zorder=3)
ax.scatter(phantoms, np.zeros(4), s=60, c='gray', marker='x', label='Phantoms', zorder=3)
ax.axvline(true_outcome, color='green', linestyle='--', linewidth=2, label=f'Outcome: {true_outcome}')
ax.axvline(voter_peaks[0], color='blue', linestyle=':', alpha=0.5)
ax.set_xlim(-1, 11)
ax.set_ylim(-0.5, 0.5)
ax.set_xlabel('Alternative space')
ax.set_title('Truth-telling: Voter 1 reports peak = 2.0')
ax.legend(loc='upper right', fontsize=8)
ax.set_yticks([])

# Right plot: Voter 1 misreports (tries to pull outcome left)
ax = axes[1]
manipulated_peaks = voter_peaks.copy()
manipulated_peaks[0] = 0.0  # Voter 1 reports 0 instead of 2
manip_outcome = generalized_median(manipulated_peaks, phantoms)
ax.scatter(manipulated_peaks, np.zeros(5), s=100, c='blue', label='Reported peaks', zorder=3)
```

```
ax.scatter([voter_peaks[0]], [0], s=100, c='red', marker='o', facecolors='none',
 ↪  linewidths=2, label='True peak', zorder=4)
ax.scatter(phantoms, np.zeros(4), s=60, c='gray', marker='x', label='Phantoms',
 ↪  zorder=3)
ax.axvline(manip_outcome, color='green', linestyle='--', linewidth=2, label=f'Outcome:
 ↪  {manip_outcome}')
ax.set_xlim(-1, 11)
ax.set_ylim(-0.5, 0.5)
ax.set_xlabel('Alternative space')
ax.set_title('Manipulation attempt: Voter 1 reports 0.0')
ax.legend(loc='upper right', fontsize=8)
ax.set_yticks([])

plt.tight_layout()
plt.show()

# Check: Is the manipulated outcome closer to voter 1's true peak?
print(f"\nVoter 1's true peak: {voter_peaks[0]}")
print(f"Outcome with truth-telling: {true_outcome} (distance: {abs(true_outcome -
 ↪  voter_peaks[0]):.1f})")
print(f"Outcome with manipulation: {manip_outcome} (distance: {abs(manip_outcome -
 ↪  voter_peaks[0]):.1f})")
print("Manipulation did not help voter 1!" if abs(manip_outcome - voter_peaks[0]) >=
 ↪  abs(true_outcome - voter_peaks[0]) else "Manipulation helped!")
```

5.3 Scoring Rules and the DPO-Borda Connection

Another way to escape Arrow's impossibility is to relax the Independence of Irrelevant Alternatives (IIA) axiom. The **Borda count** is a classic scoring rule that does exactly this.

5.3.1 The Borda Count

DEFINITION: BORDA COUNT / SOFT COPELAND SCORE

For a preference profile, the **Borda score** (or soft Copeland score) of alternative $y \in Y$ equals the expected number of alternatives ranked below y by a randomly chosen voter. Equivalently, it counts the number of pairwise "matches" that y wins against other alternatives:

$$\text{Borda}(y) = \sum_{i=1}^{n} |\{y' \neq y : y \succ_i y'\}| = \sum_{y' \neq y} |\{i : y \succ_i y'\}| \tag{5.1}$$

The **Borda winner** is the alternative with the maximum Borda score.

The Borda count satisfies many desirable properties but violates IIA. However, we can define a weaker version of IIA that Borda *does* satisfy.

DEFINITION: MODIFIED IIA (IIA')

Fix alternatives $y, y' \in Y$. If two preference profiles have, for every agent: 1. The same pairwise ordering of y vs. y', AND 2. The same *number* of alternatives strictly between y and y'

then it should not be the case that $f(\succ) = y$ and $f(\succ') = y'$.

This relaxation allows the social choice to depend on *how far apart* two alternatives are in each voter's ranking, not just their relative order.

THEOREM: PROPERTIES OF BORDA COUNT

The Borda count satisfies: - **Unrestricted domain (U) - Pareto efficiency (PE) - Non-dictatorship (ND) - Modified IIA (IIA')**

By relaxing IIA to IIA', we escape Arrow's impossibility.

5.3.2 Connection to Direct Preference Optimization (DPO)

A remarkable result connects the Borda count to modern RLHF methods, specifically Direct Preference Optimization (DPO).

THEOREM: DPO-BORDA EQUIVALENCE

Assume that responses y, y' are drawn from a reference policy $\pi_{\text{ref}}(\cdot|x)$. Define the π**-weighted Borda winner** as:

$$y^* = \arg\max_{y \in Y} \mathbb{E}_{y' \sim \pi_{\text{ref}}(\cdot|x)} [\mathbf{1}[y \succ y' \mid x]] \tag{5.2}$$

Consider DPO training with reference policy π_{ref}. Then the DPO-optimal policy π_{DPO} satisfies:

$$\frac{\pi_{\text{DPO}}(y|x)}{\pi_{\text{ref}}(y|x)} \propto (\text{weighted Borda score of } y)$$

In other words, **DPO finds a policy that upweights responses proportionally to their Borda scores.** This provides a social choice interpretation of what DPO is doing: it aggregates pairwise preferences by finding the response that would win the most head-to-head matchups against alternatives drawn from the reference policy.

WHEN DPO DOES NOT FIND THE BORDA WINNER

The DPO-Borda equivalence assumes that comparison pairs are **sampled uniformly** from the reference policy and that the Bradley-Terry model is correctly specified. In practice, these assumptions often fail: (1) annotation pipelines may oversample certain response types (e.g., longer or more controversial outputs), creating **non-uniform pair sampling** that changes the implicit social choice rule; (2) different annotator pools have different preference distributions, so the effective Borda score depends on *who* annotates *which* pairs; (3) if the Bradley-

> Terry model is misspecified (e.g., preferences are intransitive or context-dependent), the DPO objective no longer corresponds to any clean social choice rule. Practitioners should be aware that the Borda interpretation provides useful intuition but may not hold exactly in deployed systems.

The following code demonstrates the DPO-Borda connection empirically: we compute the Borda scores from pairwise preferences and verify that DPO's implicit reward ranking matches the Borda ranking.

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Simulate pairwise preferences from multiple annotators
n_responses = 5
n_annotators = 50

# True quality of each response (unknown in practice)
true_quality = np.array([0.2, 0.8, 1.5, 0.5, 1.0])

# Generate pairwise comparisons via Bradley-Terry
# Each annotator compares all pairs
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Collect pairwise win counts
wins = np.zeros((n_responses, n_responses))
for ann in range(n_annotators):
    # Each annotator has slightly noisy perception of quality
    perceived = true_quality + np.random.normal(0, 0.3, n_responses)
    for i in range(n_responses):
        for j in range(i+1, n_responses):
            p_i_wins = sigmoid(perceived[i] - perceived[j])
            if np.random.random() < p_i_wins:
                wins[i, j] += 1
            else:
                wins[j, i] += 1

# Compute Borda scores (= total pairwise wins)
borda_scores = wins.sum(axis=1)
n_comparisons = wins.sum(axis=1) + wins.sum(axis=0)  # total comparisons per response

# Compute DPO-implied reward: beta * log(pi/pi_ref) is proportional to Borda score
# Under uniform reference policy, DPO upweights proportionally to win rate
win_rates = borda_scores / n_comparisons
dpo_reward = np.log(win_rates / (1 - win_rates + 1e-8))  # log-odds  Borda ranking

```

```
# Visualize
fig, axes = plt.subplots(1, 3, figsize=(10, 3.5))
labels = [f'y{i+1}' for i in range(n_responses)]

# Left: Borda scores
ax = axes[0]
colors = plt.cm.viridis(np.linspace(0.2, 0.8, n_responses))
order = np.argsort(borda_scores)
ax.barh(range(n_responses), borda_scores[order], color=colors[order], alpha=0.8)
ax.set_yticks(range(n_responses))
ax.set_yticklabels([labels[i] for i in order])
ax.set_xlabel('Borda score (total wins)')
ax.set_title('Borda ranking')
ax.grid(True, alpha=0.3, axis='x')

# Middle: True quality vs Borda score
ax = axes[1]
ax.scatter(true_quality, borda_scores, s=80, color='steelblue', zorder=3)
for i in range(n_responses):
    ax.annotate(labels[i], (true_quality[i], borda_scores[i]),
                textcoords="offset points", xytext=(8, 0), fontsize=10)
corr = np.corrcoef(true_quality, borda_scores)[0, 1]
ax.set_xlabel('True quality')
ax.set_ylabel('Borda score')
ax.set_title(f'Quality vs Borda (r={corr:.2f})')
ax.grid(True, alpha=0.3)

# Right: Pairwise win matrix
ax = axes[2]
win_fractions = wins / (wins + wins.T + 1e-8)
im = ax.imshow(win_fractions, cmap='RdYlGn', vmin=0, vmax=1)
ax.set_xticks(range(n_responses))
ax.set_yticks(range(n_responses))
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)
ax.set_xlabel('Loses to →')
ax.set_ylabel('← Beats')
ax.set_title('Pairwise win rates')
plt.colorbar(im, ax=ax, shrink=0.8)

plt.tight_layout()
plt.show()

# Rankings comparison
borda_ranking = np.argsort(-borda_scores)
quality_ranking = np.argsort(-true_quality)
print("Borda ranking: ", " > ".join(labels[i] for i in borda_ranking))
print("True ranking:  ", " > ".join(labels[i] for i in quality_ranking))
```

```
print(f"\nDPO's implicit aggregation recovers the Borda winner:
    {labels[borda_ranking[0]]}")
print(f"This is the response that wins the most head-to-head matchups.")
```

PROOF SKETCH

Write $\hat{\pi} = \pi^*/\pi_{\text{ref}}$ for the density ratio. Let $\bar{\sigma}(\Delta r^*(x, y', y)) = \mathbb{P}(y \succ y'|x; r^*)$ denote the Bradley-Terry probability under the true reward r^*.

The DPO loss is:

$$\mathcal{L}_{\text{DPO}}(\pi) = -\mathbb{E}_{x,y,y'}\left[\bar{\sigma}(\Delta r^*) \cdot \log\sigma(\beta \log \frac{\hat{\pi}(y'|x)}{\hat{\pi}(y|x)}) + (1 - \bar{\sigma}(\Delta r^*)) \cdot \log\left(1 - \sigma(\beta \log \frac{\hat{\pi}(y|x)}{\hat{\pi}(y'|x)})\right)\right]$$

Taking the gradient with respect to $\hat{\pi}(y|x)$ and setting to zero:

$$\mathbb{E}_{y'\sim\hat{\pi}}\left[\sigma(\beta \log \frac{\pi(y|x)}{\pi(y'|x)})\right] = \underbrace{\mathbb{E}_{y'\sim\mathcal{D}(\cdot|x)}[\bar{\sigma}(\Delta r^*(x, y', y))]}_{\text{Borda score of } y}$$

The right-hand side is exactly the expected win rate of y against a random alternative—i.e., its Borda score.

5.4 Multi-Issue Voting and Separable Preferences

Real-world decisions often involve multiple independent issues. For example, an AI system might need to optimize for helpfulness, harmlessness, *and* honesty simultaneously. How should we aggregate preferences when there are multiple dimensions?

5.4.1 Voting by Committees

DEFINITION: VOTING BY COMMITTEES

Let $K = \{1, \ldots, k\}$ be a set of "objects" (issues), and let $Y = 2^K$ be the set of all subsets (possible outcomes include any combination of objects). A voting scheme $f : \mathcal{P}^n \to 2^K$ is **voting by committees** if for each object $x \in K$, there exists a committee $C_x = (N, W_x)$ where W_x is a family of winning coalitions, such that:

– The outcome includes object x if and only if $\{i : x \in B(\succ_i)\} \in W_x$

where $B(\succ_i)$ is voter i's top-ranked subset.

5.4.2 Separable Preferences

When can we decompose multi-issue decisions into independent single-issue decisions?

DEFINITION: SEPARABLE PREFERENCES

A preference $\succ$ on 2^K is **separable** if for all $A \subseteq K$ and $x \notin A$:

$$A \cup \{x\} \succ A \quad \iff \quad x \in G(\succ) \tag{5.3}$$

where $G(\succ) = \{x \in K : \{x\} \succ \emptyset\}$ is the set of objects the voter considers "good" in isolation.

Separability means that whether you want to add object x to a bundle doesn't depend on what's already in the bundle—it only depends on whether x is intrinsically good.

CHARACTERIZATION ON SEPARABLE DOMAINS

Theorem: A voting scheme satisfies surjectivity (S), strategy-proofness (SP), and separability (SEP) if and only if f is voting by committees.

Note: Voting by committees generally does *not* satisfy Pareto efficiency.

Application to RLHF: When training language models, we often want to optimize for multiple criteria (helpful, harmless, honest). If annotator preferences are separable across these criteria, we can aggregate each criterion independently. But preferences are often *not* separable—for instance, a highly helpful response might necessarily involve some risk of harm, creating dependencies between criteria.

5.5 *Nosy Preferences and the Liberal Paradox*

A crucial distinction in preference aggregation is between *private* preferences (caring only about your own outcomes) and *nosy* preferences (caring about what others receive).

5.5.1 *Private vs. Nosy Preferences*

DEFINITION: NOSY PREFERENCES

A preference is **nosy** if the individual cares about outcomes affecting others, not just themselves. Examples include:

- **Safety concerns**: Not wanting others to see bomb-building instructions
- **Privacy violations**: Wanting to prevent disclosure of others' personal data
- **Content moderation**: Preferring that certain content not be shown to anyone
- **Paternalism**: Wanting to protect others from choices that might harm them
- **Fairness**: Caring that others receive equitable treatment
- **Psychological harm**: Preferring that AI systems not deceive users

A preference is **private** if the individual only cares about their own allocation or experience.

Most classical social choice theory assumes private preferences. But in AI systems, especially content moderation and alignment, nosy preferences are ubiquitous.

5.5.2 Sen's Liberal Paradox

When preferences are nosy, even seemingly weak requirements can conflict.

! Theorem: Impossibility of the Paretian Liberal (Sen, 1970)

Consider a society where each individual has a "personal sphere"—a set of choices over which they should have autonomy. The following properties are inconsistent:

1. **Minimal Liberalism**: Each individual is decisive over at least one pair of alternatives in their personal sphere.
2. **Pareto Efficiency**: If everyone prefers x to y, then society should choose x over y.
3. **Unrestricted Domain**: Any preference profile is admissible.

Classic Example: The Prude and the Book

Consider two individuals, Prude and Lewd, and a controversial book. The alternatives are: - a: Prude reads the book - b: Lewd reads the book - c: No one reads the book

Prude's preferences: $c \succ_P a \succ_P b$ (prefers no one reads it, but would rather read it themselves than let Lewd read it—a nosy preference!)

Lewd's preferences: $a \succ_L b \succ_L c$ (wants Prude to read it, then themselves, rather than no one reading it—also nosy!)

- By Minimal Liberalism, Prude should decide between a and c (their personal reading choice). Prude prefers c.
- By Minimal Liberalism, Lewd should decide between b and c (their personal reading choice). Lewd prefers b.
- But both prefer a to b! By Pareto, society should prefer a to b.

This creates a cycle: c beats a (Prude's liberty), b beats c (Lewd's liberty), but a beats b (Pareto). There is no consistent social choice.

Implications for AI Alignment: The Liberal Paradox suggests that when different stakeholders have nosy preferences about AI behavior, there may be no way to respect both individual autonomy and collective efficiency. Content moderation decisions often involve exactly this tension—one user's preference for free expression conflicts with another's preference for a safe environment.

5.6 Case Study: Community Notes

Community Notes (formerly Birdwatch) is a real-world system for aggregating preferences about content helpfulness that addresses some of the challenges discussed above.

5.6.1 The Community Notes Model

Community Notes uses a factor model to identify "bridging" notes—notes that receive positive ratings from users across ideological divides:

$$u(y; \alpha, p, \varepsilon) = \mu + \alpha_j + \beta_j + p^\top q_j + \varepsilon_j \tag{5.4}$$

where: - μ is a global intercept - α_j is a rater intercept (some raters are generally more positive) - β_j is a note intercept (some notes are generally better) - p and q_j are k-dimensional factor vectors capturing ideological alignment - ε_j is residual noise

The key insight is that β_j captures note quality *after controlling for ideological agreement*. A note is selected if $\beta_j \geq c$ for some threshold c. This means a note must be rated positively by users who *disagree* ideologically, not just by an ideological majority.

5.6.2 Why Bridging Works

Traditional majority voting would select notes favored by the largest ideological group. Community Notes instead identifies notes with broad cross-partisan appeal. This is analogous to finding a Condorcet winner that beats all alternatives in pairwise comparisons across different subgroups.

The factor model approach has connections to: - **Collaborative filtering**: The $p^\top q_j$ term is similar to matrix factorization in recommender systems - **Item response theory**: The model resembles the Rasch model from Chapter 1, extended with latent factors - **Jury theorems**: Under certain conditions, diverse juries aggregate to correct answers better than homogeneous majorities

5.6.3 Implementing Community Notes

The following code implements a simplified Community Notes model. We generate synthetic ratings where users have ideological positions and notes have both quality (what we want to estimate) and ideological slant (what we want to control for). The key question is: can we recover true note quality by factoring out ideological agreement?

```
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Generate synthetic data
n_users = 100
n_notes = 30
k_factors = 1  # 1D ideological spectrum
```

```

# Users: ideological position on [-1, 1]
user_ideology = np.random.uniform(-1, 1, n_users)

# Notes: true quality + ideological slant
note_quality = np.random.normal(0, 1, n_notes)        # beta_j: what we want to estimate
note_slant = np.random.uniform(-1, 1, n_notes)         # q_j: ideological lean
rater_bias = np.random.normal(0, 0.3, n_users)         # alpha_i: some raters are harsher

# Generate ratings: r_ij = mu + alpha_i + beta_j + p_i * q_j + noise
mu = 3.0  # global intercept (on a 1-5 scale)
noise_std = 0.5

# Not all users rate all notes (sparse observations)
observe_prob = 0.4
observed = np.random.random((n_users, n_notes)) < observe_prob

ratings = np.full((n_users, n_notes), np.nan)
for i in range(n_users):
    for j in range(n_notes):
        if observed[i, j]:
            ratings[i, j] = (mu + rater_bias[i] + note_quality[j]
                             + user_ideology[i] * note_slant[j]
                             + np.random.normal(0, noise_std))

print(f"Generated {np.sum(observed)} ratings from {n_users} users on {n_notes} notes")
print(f"Observation density: {np.mean(observed):.1%}")

# Method 1: Naive average (ignoring ideology)
naive_scores = np.nanmean(ratings, axis=0)

# Method 2: Bridging model - control for ideology via regression
# For each note j, fit: r_ij = a + b * p_i + eps
# The intercept a estimates quality controlling for ideology
bridging_scores = np.zeros(n_notes)
for j in range(n_notes):
    mask = observed[:, j]
    if mask.sum() < 3:
        bridging_scores[j] = np.nan
        continue
    r = ratings[mask, j]
    p = user_ideology[mask]
    # OLS: r = a + b*p
    X = np.column_stack([np.ones(mask.sum()), p])
    beta_hat = np.linalg.lstsq(X, r, rcond=None)[0]
    bridging_scores[j] = beta_hat[0] - mu  # subtract global mean to get quality

```

```
# Compare both methods to ground truth
fig, axes = plt.subplots(1, 3, figsize=(10, 3.5))

# Left: Naive vs true quality
ax = axes[0]
ax.scatter(note_quality, naive_scores - mu, alpha=0.6, s=40, color='steelblue')
corr_naive = np.corrcoef(note_quality, naive_scores - mu)[0, 1]
ax.plot([-3, 3], [-3, 3], 'k--', alpha=0.3)
ax.set_xlabel('True quality')
ax.set_ylabel('Estimated quality')
ax.set_title(f'Naive average (r={corr_naive:.2f})')
ax.grid(True, alpha=0.3)

# Middle: Bridging model vs true quality
ax = axes[1]
valid = ~np.isnan(bridging_scores)
ax.scatter(note_quality[valid], bridging_scores[valid], alpha=0.6, s=40,
 ↪  color='coral')
corr_bridge = np.corrcoef(note_quality[valid], bridging_scores[valid])[0, 1]
ax.plot([-3, 3], [-3, 3], 'k--', alpha=0.3)
ax.set_xlabel('True quality')
ax.set_ylabel('Estimated quality')
ax.set_title(f'Bridging model (r={corr_bridge:.2f})')
ax.grid(True, alpha=0.3)

# Right: Show how bridging identifies cross-partisan notes
ax = axes[2]
# Color notes by their slant
colors = plt.cm.RdBu(0.5 + note_slant[valid] / 2)
scatter = ax.scatter(note_quality[valid], bridging_scores[valid],
                     c=note_slant[valid], cmap='RdBu', s=50, alpha=0.7,
                     vmin=-1, vmax=1)
ax.plot([-3, 3], [-3, 3], 'k--', alpha=0.3)
ax.set_xlabel('True quality')
ax.set_ylabel('Bridging score')
ax.set_title('Colored by ideological slant')
plt.colorbar(scatter, ax=ax, label='Note slant')
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Analysis
print(f"\nCorrelation with true quality:")
print(f"  Naive average:   {corr_naive:.3f}")
print(f"  Bridging model:  {corr_bridge:.3f}")
print(f"\nThe bridging model better recovers true quality by controlling")
print(f"for ideological agreement between users and notes.")

```

```
# Show which notes would be selected under each method
threshold = 0.5
naive_selected = set(np.where((naive_scores - mu) > threshold)[0])
bridge_selected = set(np.where(bridging_scores > threshold)[0])
true_good = set(np.where(note_quality > threshold)[0])

print(f"\nNotes selected as 'high quality' (threshold={threshold}):")
print(f"  Ground truth:    {len(true_good)} notes")
print(f"  Naive method:    {len(naive_selected)} notes  (precision:
 ↪  {len(naive_selected & true_good)/max(len(naive_selected),1):.0%})")
print(f"  Bridging method: {len(bridge_selected)} notes (precision:
 ↪  {len(bridge_selected & true_good)/max(len(bridge_selected),1):.0%})")
```

The bridging model's advantage is most pronounced for **polarizing notes**—notes with strong ideological slant that would be overrated by ideological allies and underrated by opponents under naive averaging. By controlling for the $p_i \cdot q_j$ interaction term, the model isolates genuine quality from ideological agreement, effectively identifying notes that earn cross-partisan respect.

5.7 The Inversion Problem: Behavior vs. Preferences

A fundamental challenge in preference learning is the **inversion problem**: observed behavior does not necessarily reveal underlying preferences or utility.

5.7.1 Behavior □ Mental State

! THE INVERSION PROBLEM

Standard revealed preference theory assumes that choices reveal preferences: if someone chooses x over y, they must prefer x to y. But this assumption can fail when:

1. **Habit formation**: Repeated behavior creates habits that persist even when preferences change
2. **Cognitive limitations**: Fatigue, distraction, or bounded rationality lead to suboptimal choices
3. **Context effects**: The same underlying preference produces different behaviors in different contexts
4. **Strategic behavior**: People may choose strategically rather than according to true preferences

Example: The Doritos Problem

Consider a smart pantry system that observes eating behavior. A user consistently chooses Doritos when offered. The system infers: "User prefers Doritos." But the user might actually prefer healthier options—they just succumb to availability and habit when Doritos are present. Optimizing for observed "preferences" (engagement) may not optimize for true welfare.

Example: Doctor Fatigue

Studies show that doctors' prescribing patterns change systematically throughout the day as fatigue accumulates. Late-afternoon decisions may reflect fatigue more than medical judgment. A preference learning system that treats all decisions equally would encode fatigue-induced biases as "preferences."

5.7.2 Implications for RLHF

The inversion problem has direct implications for training AI systems from human feedback:

1. **Annotator fatigue**: Quality of preference labels may degrade over long annotation sessions
2. **Engagement vs. satisfaction**: Optimizing for clicks or watch time may not optimize for user welfare
3. **Context-dependent feedback**: The same annotator might give different feedback depending on their mood, prior examples, or framing
4. **Strategic annotation**: If annotators believe certain patterns lead to better outcomes for them, they may annotate strategically

Potential solutions: - Weight annotations by estimated quality or consistency - Use deliberation or discussion before labeling to reduce noise - Explicitly model annotator state (fatigue, expertise) as latent variables - Collect meta-feedback about label confidence

5.8 Privacy and Personalization

Preference learning inherently involves collecting personal data. This creates tension between privacy protection and personalization quality.

5.8.1 Contextual Integrity

Helen Nissenbaum's **Contextual Integrity** framework provides a nuanced approach to privacy that goes beyond simple "consent" models.

Contextual Integrity Framework

Privacy is preserved when information flows match context-specific norms. Five parameters determine appropriateness:

1. **Sender**: Who is sharing the information
2. **Subject**: Whose information is being shared
3. **Recipient**: Who receives the information
4. **Data Type**: What kind of information
5. **Transmission Principle**: What rules govern further use

A privacy violation occurs when information flows in ways that violate contextual norms, even if the subject "consented."

Example: Heart Rate Data

Consider a fitness tracker that collects heart rate data: - **Acceptable flow**: Sending to a running coach for training optimization - **Unacceptable flow**: Sending to an advertising network for targeted ads - **The difference**: The transmission principle (coaching vs. advertising) violates the user's expectations about the fitness context

5.8.2 Differential Privacy Limitations

Differential privacy (DP) provides formal guarantees that algorithms don't reveal individual information. However, DP has fundamental limitations for preference learning:

1. **Personalization requires individual data**: By definition, DP prevents using individual data for personalization
2. **Trade-off is inherent**: Stronger privacy guarantees mean less accurate preference models
3. **"Persuasive" use of DP**: Some systems claim DP protection with parameters so weak they provide little actual privacy

Contextual integrity offers a middle ground: allow data use that matches user expectations (e.g., personalization within a service) while preventing unexpected flows (e.g., selling data to third parties).

5.9 Paternalism in AI Systems

When should an AI system override a user's stated preferences?

5.9.1 Paternalism vs. Nosy Preferences

Paternalism differs from nosy preferences: - **Nosy preferences**: Caring about others' choices for *your own* sake - **Paternalism**: Overriding others' choices for *their* sake

5.9.2 When is Paternalism Justified?

Several conditions might justify paternalistic intervention:

1. **Information asymmetry**: The system has information the user lacks (e.g., long-term health effects)
2. **Cognitive limitations**: The user is impaired (fatigue, addiction, cognitive decline)
3. **Protection of future self**: The user's current choice harms their future self (e.g., saving for retirement)
4. **Irreversible harm**: The consequences of a choice are severe and irreversible

5.9.3 Design Principles

AI systems that exercise any form of paternalism should:

1. **Be transparent**: Users should know when and why their preferences are being overridden
2. **Allow override**: Users should be able to insist on their original choice
3. **Minimize interference**: The lightest intervention that achieves the goal should be used
4. **Justify interventions**: Each paternalistic choice should have a clear rationale
5. **Update based on feedback**: Systems should learn when interventions are welcomed vs. resented

Example: AI Assistants Refusing Requests

When a user asks an AI assistant for harmful information (e.g., how to synthesize drugs), the refusal is paternalistic if motivated by protecting the user, or nosy if motivated by protecting others. In practice, refusals often serve both purposes.

5.10 Mechanism Design

While voting rules aggregate ordinal rank preferences to select a social outcome, another class of preference aggregation occurs in economic settings like auctions and general mechanism design. Here individuals reveal their valuations (numerical utilities) for outcomes, and the mechanism chooses an outcome (such as an allocation of goods) and possibly payments. Mechanism design asks: how can we design rules so that rational agents, acting in their own interest, end up revealing information that leads to a socially desirable outcome? A central concept is incentive compatibility – the mechanism should be designed so that each participant's best strategy is to act according to their true preferences (e.g. bid their true value). In this section, we focus on auctions as a prime example of preference aggregation with money, and highlight classical results including Vickrey–Clarke–Groves (VCG) mechanisms and Myerson's optimal auction.

Consider a single-item auction with one item for sale and n bidders. Bidder i has a private valuation v_i for the item (how much the item is worth to them). Each bidder's goal is to maximize their own utility, defined as $v_i - p_i$ if they win and pay price p_i, or 0 if they do not win (assuming quasilinear utility where money is the transferable utility). The auction's task is to allocate the item to one of the bidders and possibly determine payments. We can think of an auction as a mechanism that asks each bidder for a "message" (typically a bid representing how much they are willing to pay), then selects a winner and a price based on the bids. A key objective might be social welfare maximization – allocate the item to the bidder who values it most (maximizing v_i of the winner). Another possible objective is revenue maximization for the seller – choose the allocation and price to maximize the seller's expected payment.

A classic result in auction theory is that to maximize social welfare in a single-item private-value setting, one should award the item to the highest valuer – and this can be done in an incentive-compatible way by using a second-price auction. The Vickrey second-price auction works as follows: (1) All bidders submit sealed bids $b_1, b_2, \ldots, b_n$. (2) The bidder with the

highest bid wins the item. (3) The price paid by the winner is the second-highest bid. For example, if the bids are $(2, 6, 4, 1)$ (in some currency units), the highest bid is 6 (by bidder 2, say) and the second-highest is 4. Bidder 2 wins the item and pays 4.

Under this mechanism, it turns out that bidding truthfully $b_i = v_i$ is a dominant strategy for each bidder. In other words, the auction is dominant-strategy incentive compatible (DSIC): no matter what others do, a bidder maximizes their expected utility by reporting their true valuation. The intuition is as follows. If bidder i bids lower than their true value (i.e. $b_i < v_i$), and if their true value was actually the highest, they risk losing the item even though they value it more than the price they would have paid – a missed opportunity for positive utility. Bidding higher than their value ($b_i > v_i$) cannot help them win in any situation where bidding truthfully wouldn't (it could only make a difference if their true v_i wasn't the highest but they tried to win anyway); and if they do win with an inflated bid, they might end up paying the second-highest bid which could be above their true value, yielding negative utility. By bidding exactly v_i, if they win, it means all other bids were lower, so v_i is at least as high as the second-highest bid p they pay – guaranteeing non-negative utility $v_i - p \geq 0$. If they lose, it means someone else had a higher bid (hence higher value, if others are truthful), so bidder i wouldn't have gained anyway. This argument, made rigorous by Vickrey (Vickrey 1961), establishes that truth-telling is a dominant strategy in the second-price auction. As a consequence, when everyone bids truthfully, the item is allocated to the bidder with the highest v_i, achieving maximum social surplus (allocative efficiency). The second-price auction is thus an elegant mechanism that aligns individual incentives with social welfare maximization.

It is worth contrasting this with a first-price auction, where the winner pays their own bid. In a first-price auction, bidders have an incentive to bid below their true value (to avoid the winner's curse of paying too much), in a Nash equilibrium that involves bid shading. The first-price auction can still allocate to the highest valuer in equilibrium, but only through strategic behavior (and it is not DSIC). By charging the second-highest bid, the Vickrey auction removes the incentive to shade bids, since the price does not directly depend on one's own bid beyond the fact of winning or losing.

So far, we discussed auctions aimed at maximizing social welfare. In many cases, the auctioneer (seller) is interested in maximizing revenue. A foundational result by Roger Myerson (1981) provides a characterization of optimal auctions (those that maximize the seller's expected revenue) for single-item settings under certain assumptions (Myerson 1981). The problem can be formulated as follows: suppose each bidder's private value v_i is drawn independently from a known distribution F_i (for simplicity, assume identical distribution F for all bidders, i.i.d.). We seek a mechanism (allocation rule and payment rule) that maximizes the seller's expected payment, subject to incentive compatibility and individual rationality (participants should not expect negative utility from truthful participation).

Myerson's theorem states that the optimal auction in such a setting is a threshold auction characterized by virtual valuations. Define the virtual value for a bidder with value v as $\varphi(v) = v - \frac{1-F(v)}{f(v)}$, where f is the probability density function of F (assuming it is continuous). An assumption called regularity (which holds for many distributions) is that $\varphi(v)$ is non-decreasing in v. Myerson showed that the revenue-maximizing strategy is: treat $\varphi(v)$ as

the effective "score" of a bid, allocate the item to the bidder with the highest non-negative virtual value (if all virtual values are negative, allocate to no one), and charge them the smallest value they could have such that they would still win (the payment is essentially the critical bid where φ of that bid equals the second-highest virtual value or the zero cutoff). In practice, for i.i.d. bidders, this reduces to: there is an optimal reserve price r such that you sell to the highest bidder if and only if their bid $b_{\max} \geq r$; if sold, the price is the max of the second-highest bid and r.

In the case of n bidders with values i.i.d. uniform on $[0, 1]$ (which is a regular distribution), one can compute the optimal reserve price. The virtual value function for uniform $[0, 1]$ is $\varphi(v) = v - \frac{1-v}{1} = 2v - 1$. Setting $\varphi(v) \geq 0$ gives $v \geq 0.5$. So Myerson's mechanism says: don't sell the item if all bids are below 0.5; otherwise, sell to the highest bidder at at least 0.5. This is exactly a second-price auction with a reserve of $r = 0.5$. Our earlier example implicitly demonstrated this: with two uniform(0,1) bidders, the optimal auction sets a reserve price of 0.5 and yields a certain expected revenue. We can break down the cases: - With probability $1/4$, both bidders have values below 0.5 (each below 0.5 with probability 1/2), in which case nobody wins and revenue is 0. - With probability $1/4$, both bidders have $v > 0.5$. In this case, the second-price auction with reserve will sell to the highest bidder at the max of the second-highest value and 0.5. Given both $v_1, v_2 > 0.5$, the expected second-highest value (conditional on both >0.5) is $\frac{2}{3}$ (in fact, the order statistics of two uniforms on [0.5,1] give mean of min = 2/3). So in this case the expected price is the second-highest value (since that will exceed 0.5), about 0.667. - With probability $1/2$, one bidder is above 0.5 and the other below. In that case, the one above 0.5 wins at price equal to the reserve 0.5 (since the second-highest bid is the reserve).

Taking the expectation, the seller's expected revenue is $0 * (1/4) + (2/3) * (1/4) + (1/2 * 1/2) = 0 + 1/6 + 1/4 = 5/12 \approx 0.417$. This is higher than the expected revenue without a reserve. In fact, without a reserve (just a plain second-price with two bidders uniform [0,1]), one can compute the expected revenue is $1/3 \approx 0.333$ (the second order statistic's expectation). Thus, the reserve has increased revenue. Myerson's theory tells us that indeed the second-price auction with an optimally chosen reserve maximizes revenue among all DSIC mechanisms for this setting. A notable special case result is that when bidder distributions are i.i.d. and regular, an optimal auction is essentially "allocatively efficient with a reserve price" – i.e. aside from possibly excluding low-value bidders via a reserve, it allocates to the highest remaining bid.

Myerson's work also highlighted the gap between revenue maximization and welfare maximization. The price of optimality (in revenue) is that the seller might sometimes forego efficient allocation (e.g. not selling despite a willing buyer, in order to preserve a high reserve price strategy). In contrast, Vickrey's auction always allocates efficiently but may not maximize revenue.

An interesting insight in auction theory is that increasing competition can yield more revenue than fine-tuning the auction mechanism. The Bulow–Klemperer theorem (Bulow and Klemperer 1996) demonstrates that, under certain regularity assumptions, a simple welfare-maximizing auction with one extra bidder outperforms the optimal auction with fewer bidders. Specifically, for i.i.d. bidders with a regular distribution F, the expected revenue of

a second-price auction with $n+1$ bidders is at least as high as the expected revenue of the Myerson-optimal auction with n bidders. In formula form:

$$\mathbb{E}_{v_1,\dots,v_{n+1}\sim F}[\text{Rev}^{(\text{second-price})}(n+1 \text{ bidders})] \geq \mathbb{E}_{v_1,\dots,v_n\sim F}[\text{Rev}^{(\text{optimal})}(n \text{ bidders})]\,. \quad (5.5)$$

This result suggests that, in practice, having more participants (competition) is often more valuable than exploiting detailed knowledge of bidder distributions. As a corollary, a policy recommendation is that a seller is usually better off using a simple auction design (like a Vickrey auction or other transparent mechanism) and putting effort into attracting more bidders, rather than using a complex optimal mechanism that might discourage participation.

5.10.0.1 The VCG Mechanism

Vickrey's second-price auction can be generalized to multiple items and more complex outcomes by the Vickrey–Clarke–Groves (VCG) mechanism. The VCG mechanism is a cornerstone of mechanism design that provides a general solution for implementing socially efficient outcomes (maximizing total stated value) in dominant strategies, for a broad class of problems. It works for any scenario where agents have quasilinear utilities and we want to maximize the sum of valuations.

In a general mechanism design setting, let Ω be the set of possible outcomes. Each agent i has a private valuation function $v_i(\omega)$ for outcomes $\omega \in \Omega$ (the amount of utility, in money terms, that i gets from outcome ω). Agents report bids $b_i(\omega)$ (which we hope equal $v_i(\omega)$ if they are truthful). The mechanism then chooses an outcome $\omega^* \in \Omega$ to maximize the reported total value:

$$\omega^* = \arg\max_{\omega\in\Omega} \sum_{i=1}^{n} b_i(\omega)\,, \quad (5.6)$$

i.e. ω^* is the outcome that would be socially optimal if the b_i were true values. To induce truth-telling, VCG sets payments such that each agent pays the externality they impose on others by their presence. Specifically, one convenient form of the VCG payment for agent i is:

$$p_i(b) = \max_{\omega\in\Omega} \sum_{j\neq i} b_j(\omega) \;-\; \sum_{j\neq i} b_j(\omega^*)\,, \quad (5.7)$$

which can be interpreted as: what would the total value of others be if i were not present (first term, maximizing without i) minus the total value others actually get in the chosen outcome ω^*. Equivalently, we can write the payment as the agent's bid for the chosen outcome minus a rebate term:

$$p_i(b) = b_i(\omega^*) \; - \; \Big[\sum_{j=1}^{n} b_j(\omega^*) - \max_{\omega \in \Omega} \sum_{j \neq i} b_j(\omega) \Big] . \tag{5.8}$$

This formula (which in single-item auction reduces to second-price logic) ensures that each agent's net payoff is $v_i(\omega^*) - p_i = \max_\omega \sum_{j \neq i} v_j(\omega) + v_i(\omega^*) - \sum_{j \neq i} v_j(\omega^*)$. All terms except $v_i(\omega^*)$ cancel out, meaning each agent's utility equals the max total welfare of others plus their own value for the chosen outcome minus others' welfare in the chosen outcome – which does not depend on $v_i(\omega^*)$ except through the decision of ω^*. By construction, an agent cannot influence ω^* in a way that improves this expression unless it genuinely increases total welfare, so misreporting v_i cannot increase their utility. Thus truthful reporting is a dominant strategy. VCG is dominant-strategy incentive compatible (DSIC) and produces an outcome that maximizes $\sum_i v_i(\omega)$, achieving social welfare maximization.

VCG provides a powerful existence result: under broad conditions, there is a mechanism that achieves efficient allocation with truth-telling (in fact, VCG is essentially the unique one, aside from adding harmless constant transfers). However, implementing VCG in practice can be difficult. One challenge is computational: finding $\arg\max_\omega \sum_i b_i(\omega)$ can be NP-hard if Ω is a combinatorially large space (as in many combinatorial auctions). Another issue is budget balance and revenue: VCG payments might not yield any revenue to the mechanism designer in some cases (or even require subsidies in complex settings), and they can be low or zero in certain environments, which is problematic if the seller needs revenue. VCG is also vulnerable to collusion or the presence of fake identities (sybil attacks) – the mechanism assumes each participant is a separate entity; if one bidder can split into two identities, they might game the outcome.

Nonetheless, for many domains, VCG or variants have been successfully used or at least studied. Notably, combinatorial auctions (where multiple items are up for sale and bidders have valuations for bundles of items) can in theory be handled by VCG: just let Ω be all possible allocations of items to bidders, and have bidders report $b_i(S)$ for each bundle S of items. VCG would allocate the items in the way that maximizes total reported value and charge each bidder the opportunity cost their presence imposes on others. In practice, as mentioned, combinatorial auctions face exponential complexity in preference reporting (each bidder potentially has to specify a value for every subset of items) and winner determination (solving an NP-hard combinatorial optimization). Heuristic or restricted approaches (like limiting the kinds of bundles or using iterative bidding with query learning of preferences) are used to make the problem tractable. Additionally, pure VCG in combinatorial settings can have undesirable properties: for example, in some cases adding more bidders can cause VCG prices to drop to zero (the so-called "threshold problem" or revenue monotonicity failure), and bidders may collude to manipulate their bids collectively.

One high-stakes application of combinatorial auctions is spectrum auctions for selling licenses of electromagnetic spectrum to telecom companies. Governments have used multi-round combinatorial auctions to allocate spectrum, with billions of dollars at stake. Designing these auctions requires balancing efficiency with simplicity and robustness to strategic behavior. Early

spectrum auctions that used simpler formats (like sequential auctions or one-shot sealed bids for each license) ran into problems like the exposure problem – a bidder valuing a combination of items (say complementary licenses in adjacent regions) risks winning only part of the combination at a high price, which could be bad for them if the items are worth much less separately. The simultaneous multi-round auction (SMRA) was an innovation that allowed bidding on all items at once in rounds, giving bidders some price discovery to mitigate the exposure problem. Even so, strategic issues like demand reduction (bidders deliberately not bidding on too many items to keep prices low) and tacit collusion through signaling bids have been observed. These practical complications underscore that while VCG is a beautiful theoretical ideal, real-world mechanism design often involves compromises and tweaks.

5.10.1 Case Study 1: Mechanism for Peer Grading

To illustrate an application of mechanism design beyond auctions, consider a classroom setting where students grade each other's work (peer assessment). The goal is to design a system (a "mechanism") that produces fair and accurate grades while incentivizing students to put effort into grading. Jason Hartline and colleagues (2020) studied such a scenario, examining how to optimize scoring rules for peer grading (Hartline et al. 2020). In this setting, students are both agents (who might strategize to maximize their own grade or minimize their effort) and graders. The "outcome" we want is a set of final grades for students, ideally reflecting the true quality of their work.

One idea is to use proper scoring rules to evaluate the peer graders. A proper scoring rule is a concept from forecast evaluation that gives highest expected score for truthful reporting of probabilities. In peer grading, one might try to reward students based on how close their grading is to some ground truth or to the TA's grades. However, a naive application of proper scoring can backfire. Hartline et al. observed a "lazy peer grader" problem: if students figure out that always giving an average score (say 80%) yields a decent reward under the scoring rule, they might not bother to carefully distinguish good and bad work. In one experiment, giving all peers an 80% could yield a 96% accuracy score for the grader under a certain scoring rule (Hartline et al. 2023). This clearly undermines the goal – the grader is basically cheating the system by always predicting the class average.

To combat this, the mechanism designers sought a scoring rule that maximizes the difference in reward between a diligent grading and a lazy strategy, thereby incentivizing effort. They formulated this as an optimization problem: design the reward function for peer graders such that truthful, careful grading yields a strictly higher expected score than any degenerate strategy like always giving the average. By analyzing data and grader behavior models, they adjusted the scoring rules to penalize obviously lazy patterns and reward variance when warranted. The resulting mechanism improved the accuracy of peer grading by aligning the incentives of student graders (who want a high score for their grading job) with the objective of accurate assessment. This case study highlights how ideas of incentive compatibility and mechanism design apply even in social/educational contexts: the "payments" are points towards one's own grade, and the mechanism must account for strategic behavior to ensure a reliable outcome.

In conclusion, mechanism design provides a toolkit for aggregating preferences (or signals, like grades or bids) in a principled way, by explicitly accounting for individual incentives. Whether in auctions, peer grading, or other domains, the design of rules (allocation algorithms, payment or scoring schemes) crucially determines whether people feel encouraged to be truthful or to game the system. The theories of VCG and Myerson give us optimal baselines for efficiency and revenue in auctions, while impossibility results like Gibbard–Satterthwaite warn us of the limitations in voting. Real-world implementations often have to grapple with complexity and approximate these ideals. While learning from individual human preference is a powerful approach, it too faces aggregation challenges. If the human feedback is inconsistent or if different annotators have different preferences, the reward model may end up capturing an average that satisfies no one perfectly. There is active research on scalable oversight: techniques to gather and aggregate human feedback on tasks that are too complex for any single person to evaluate reliably. This includes approaches like recursive reward modeling, iterated amplification (Christiano, Shlegeris, and Amodei 2018), and AI-assisted debate (Irving, Christiano, and Amodei 2018), where AI systems help humans provide better feedback or break down tasks. The goal of scalable oversight is to leverage human preferences and principles in guiding AI even as AI systems tackle increasingly complex or open-ended tasks, while mitigating the human burden and bias in evaluation.

In summary, preference aggregation in machine learning spans from simple models like Bradley–Terry for pairwise comparisons to elaborate RLHF pipelines for training large models. The deep mathematical foundations – whether Arrow's theorem or Myerson's auction theory – remind us that whenever we aggregate preferences or signals from multiple sources, we must consider incentive effects, fairness criteria, and the possibility of inconsistency. By combining insights from social choice, economics, and statistical learning, we aim to build AI systems that not only learn from human preferences but do so in a principled, robust, and fair manner. The next chapter will delve further into aligning AI with human values, building on the mechanisms and learning algorithms discussed here to ensure AI systems remain beneficial and in line with what people truly want.

5.10.2 Case Study 2: Incentive-Compatible Online Learning

To address this problem, we seek to create a model. We first outline the key criteria that our model must achieve. The model revolves around repeated interactions between a planner (the system) and multiple agents (the users). Each agent, upon arrival in the system, is presented with a set of available options to choose from. These options could vary widely depending on the application of the model, such as routes in a transportation network, a selection of hotels in a travel booking system, or even entertainment choices in a streaming service. The interaction process is straightforward but crucial: agents arrive, select an action from the provided options, and then report feedback based on their experience. This feedback is vital as it forms the basis upon which the planner improves and evolves its recommendations. The agents in this model are considered strategic; they aim to maximize their reward based on the information available to them. This aspect of the model acknowledges the real-world scenario where users are typically self-interested and seek to optimize their own outcomes. The planner, on the

other hand, has a broader objective. It aims to learn which alternatives are best in a given context and works to maximize the overall welfare of all agents. This involves a complex balancing act: the planner must accurately interpret feedback from a diverse set of agents, each with their own preferences and biases, and use this information to refine and improve the set of options available. The ultimate goal of the planner is to create a dynamic, responsive system that not only caters to the immediate needs of individual agents but also enhances the collective experience over time, leading to a continually improving recommendation ecosystem.

Here, we seek to address the inherent limitations faced by the planner, particularly in scenarios where monetary transfers are not an option, and the only tool at its disposal is the control over the flow of information between agents. This inquiry aims to understand the extent to which these limitations impact the planner's ability to effectively guide and influence agent behavior. A critical question is whether the planner can successfully induce exploration among agents, especially in the absence of financial incentives. This involves investigating strategies to encourage users to try less obvious or popular options, thus broadening the scope of feedback and enhancing the system's ability to learn and identify the best alternatives. Another question is understanding the rate at which the planner learns from agent interactions. This encompasses examining how different agent incentives, their willingness to explore, and their feedback impact the speed and efficiency with which the planner can identify optimal recommendations.

The model can be extended in several directions, each raising its own set of questions.

```
1. Multiple Agents with Interconnected Payoffs: When multiple agents
arrive simultaneously, their choices and payoffs become interconnected,
resembling a game. The research question here focuses on how these
interdependencies affect individual and collective decision-making.

2. Planner with Arbitrary Objective Function: Investigating scenarios
where the planner operates under an arbitrary objective function, which
might not align with maximizing overall welfare or learning the best
alternative.

3. Observed Heterogeneity Among Agents: This involves situations where
differences among agents are observable and known, akin to contextual
bandits in machine learning. The research question revolves around how
these observable traits can be used to tailor recommendations more
effectively.

4. Unobserved Heterogeneity Among Agents: This aspect delves into
scenarios where differences among agents are not directly observable,
necessitating the use of causal inference techniques to understand and
cater to diverse user needs.
```

In our setup, there is a "planner," which aims to increase exploration, and many independent "agents," which will act selfishly (in a way that they believe will maximize their individual

reward) (Mansour, Slivkins, and Syrgkanis 2019; Mansour et al. 2021). Under our model shown in Figure 1.1, there are K possible actions that all users can take, and each action has some mean reward $\mu_i \in [0, 1]$. In addition, there is a common prior belief on each μ_i across all users.. The T agents, or users, will arrive sequentially. As the t'th user arrives, they are recommended an action I_t by the planner, which they are free to follow or not follow. After taking whichever action they choose, the user experiences some realized reward $r_i \in [0, 1]$, which is stochastic i.i.d. with mean μ_i, and reports this reward back to the planner.

So far, the model we have defined is equivalent to a multi-armed bandit model, which we have seen earlier in this chapter (1). Under this model, well-known results in economics, operations research and computer science show that $O(\sqrt{T})$ regret is achievable (Russo and Roy 2015; Auer, Cesa-Bianchi, and Fischer 2002; Lai and Robbins 1985) with algorithms such as Thompson sampling and UCB. However, our agents are strategic and aim to maximize their own rewards. If they observe the rewards gained from actions taken by other previous users, they will simply take the action they believe will yield the highest reward given the previous actions; they would prefer to benefit from exploration done by other users rather than take the risk of exploring themselves. Therefore, exploration on an individual level, which the planner would like to facilitate, is not guaranteed under this paradigm.

In light of this, we also require that our model satisfy incentive compatibility, or that taking the action recommended by the planner has an expected utility that is as high as any other action the agent could take. Formally, $\forall i : E[\mu_i | I_t = i] \geq E[\mu_{i'} | I_t = i]$. Note that this incentivizes the agents to actually take the actions recommended by the planner; if incentive compatibility is not satisfied, agents would simply ignore the planner and take whatever action they think will lead to the highest reward.

At a high level, the key to achieving incentive compatibility while still creating a policy for the planner that facilitates exploration is information asymmetry. Under this paradigm, the users only have access to their previous recommendations, actions, and rewards, and not to the recommendations, actions, and rewards of other users. Therefore, they are unsure of whether, after other users take certain actions and receive certain rewards, arms that they might have initially considered worse in practice outperform arms that they initially considered better. Only the planner has access to the previous actions and rewards of all users; the user only has access to their own recommendations and overall knowledge of the planner's policy. The main question we aim to answer for the rest of this section is, given this new constraint of incentive compatibility, is $O(\sqrt{T})$ regret still achievable? We illustrate such an algorithm in the following.

The main result here is a black-box reduction algorithm to turn any bandit algorithm into an incentive compatible one, with only a constant increase in Bayesian regret. Since, as mentioned earlier, there are bandit algorithms with $O(\sqrt{T})$ Bayesian regret, black-box reduction will also allow us to get incentive-compatible algorithms with $O(\sqrt{T})$ regret. The idea of black-box reduction will be to simulate T steps of any bandit algorithm in an incentive-compatible way in cT steps. This allows us to design incentive-compatible recommendation systems by using any bandit algorithm and then adapting it. Consider the following setting: there are two possible actions, A_1 and A_2. Assume the setting of deterministic rewards, where action 1 has reward μ_1

with prior $U[1/3, 1]$ and mean $\mathbb{E}[\mu_1] = 2/3$, and action 2 has reward μ_2 with prior $U[0, 1]$ and mean $\mathbb{E}[\mu_2] = 1/2$. Without the planner intervention and with full observability, users would simply always pick A_1, so how can the planner incentivize users to play A_2?

The key insight is going to be to hide exploration in a pool of exploitation. The users are only going to receive a recommendation from the planner, and no other observations. After deterministically recommending the action with the highest expected reward (A_1), the planner will pick one guinea pig to recommend the exploratory action of A_2. The users don't know whether they are the guinea pig, so intuitively, as long as the planner picks guinea pigs uniformly at random and at low enough frequencies, the optimal decision for the users is still to follow the planner's recommendation, even if it might go against their interest. The planner will pick the user who will be recommended the exploratory action uniformly at random from the L users that come after the first one (which deterministically gets recommended the exploitation action). Under this setting (illustrated in Figure 1.2), it is optimal for users to always follow the option that is recommended for them. More formally, if I_t is the recommendation that a user receives at time t, then we have that:

$$\begin{aligned}\mathbb{E}[\mu_1 - \mu_2 | I_t = 2] Pr[I_t = 2] = \frac{1}{L}(\mu_1 - \mu_2) \quad &\text{(Gains if you are the unlucky guinea pig)} \\ + (1 - \frac{1}{L})\mathbb{E}[\mu_1 - \mu_2 | \mu_1 < \mu_2] \times p[\mu_1 < \mu_2] \quad &\text{(Loss if you are not and } \mu_1 < \mu_2\text{)} \\ \leq 0 \quad &\end{aligned} \tag{5.9}$$

This holds when $L \geq 12$. It means that the gains from not taking the recommended action are negative, which implies that users should always take the recommendation. So far we have considered the case where rewards are deterministic, but what about stochastic rewards? We are now going to consider the case where rewards are independent and identically distributed from some distribution, and where each action A_i has some reward distribution $r_i^t \sim D_i, \mathbb{E}[r_i^t] = \mu_i$. Back to the case where there are only two actions, we are going to adapt the prior algorithm of guinea pig-picking to the stochastic reward setting. Since one reward observation is not enough to fully know μ_1 anymore, we'll instead observe the outcome of the first action M times to form a strong posterior $\mathbb{E}[\mu_1 | r_1^1, ... r_1^M]$. We can use with stochastic rewards when there are two actions. Similarly, as before, we pick one guinea pig uniformly at random from the next L users and use the reward we get as the exploratory signal. In a very similar manner, we can generalize this algorithm from always having two actions to the general multi-armed bandit problem. Now suppose we have a general multi-armed bandit algorithm A. We will wrap this algorithm around our black box reduction algorithm to make it incentive-compatible. We wrap every decision that A would make by exactly $L - 1$ recommendations of the action believed to be the best so far. This guarantees that the expected rewards for the users that are not chosen as guinea pigs are at least as good as A's reward at phase n.

5.11 Mutual Information Paradigm

In this section we discuss an influential new framework for designing peer prediction mechanisms, the Mutual Information Paradigm (MIP) introduced by Kong and Schoenebeck (Kong and Schoenebeck 2019). Traditional peer prediction approaches typically rely on scoring rules and correlation between agents' signals. However, these methods often struggle with issues like uninformed equilibria, where agents can coordinate on uninformative strategies that yield higher payoffs than truth-telling. The core idea is to reward agents based on the mutual information between their report and the reports of other agents. We consider a setting with n agents, each possessing a private signal Ψ_i drawn from some set Σ. The mechanism asks each agent to report their signal, which we denote as $\hat{\Psi}_i$. For each agent i, the mechanism randomly selects a reference agent $j \neq i$. Agent i's payment is then calculated as $MI(\hat{\Psi}_i; \hat{\Psi}_j)$ where MI is an information-monotone mutual information measure. An information-monotone MI measure must satisfy the following properties:

- Symmetry: $MI(X;Y) = MI(Y;X)$.
- Non-negativity: $MI(X;Y) \geq 0$, with equality if and only if X and Y are independent.
- Data processing inequality: For any transition probability M, if Y is independent of $M(X)$ conditioned on X, then $MI(M(X);Y) \leq MI(X;Y)$.

Two important families of mutual information measures that satisfy these properties are f-mutual information and Bregman mutual information. The f-mutual information is defined as $MI_f(X;Y) = D_f(U_{X,Y}, V_{X,Y})$, where D_f is an f-divergence, $U_{X,Y}$ is the joint distribution of X and Y, and $V_{X,Y}$ is the product of their marginal distributions. The Bregman mutual information is defined as: $BMI_{PS}(X;Y) = \mathbb{E}_X[DPS(U_{Y|X}, U_Y)]$, where D_{PS} is a Bregman divergence based on a proper scoring rule PS, $U_{Y|X}$ is the conditional distribution of Y given X, and U_Y is the marginal distribution of Y. The MIP framework can be applied in both single-question and multi-question settings. In the multi-question setting, the mechanism can estimate the mutual information empirically from multiple questions. In the single-question setting, additional techniques like asking for predictions about other agents' reports are used to estimate the mutual information. A key theoretical result of the MIP framework is that when the chosen mutual information measure is strictly information-monotone with respect to agents' priors, the resulting mechanism is both dominantly truthful and strongly truthful. This means that truth-telling is a dominant strategy for each agent and that the truth-telling equilibrium yields strictly higher payoffs than any other non-permutation strategy profile. As research continues to address practical implementation challenges of designing truthful mechanisms, MIP-based approaches have significant potential to improve preference elicitation and aggregation in real-world applications lacking verifiable ground truth.

Practical Limitations of Peer Prediction

While the MIP framework provides elegant theoretical guarantees for truthful elicitation, practical deployment faces several challenges. First, it requires a **common prior** or sufficient data to estimate the joint distribution

of reports—which may not be available in small-scale annotation tasks. Second, theoretical guarantees hold in the **multi-question** setting where mutual information can be estimated empirically; in the **single-question** setting, additional assumptions (e.g., asking agents to predict others' reports) are needed. Third, the gap between theoretical guarantees and empirical performance can be substantial: agents may not reason about dominant strategies, and the payment scheme's complexity may itself introduce noise. In annotation pipelines for RLHF, simpler quality-control mechanisms (inter-annotator agreement, gold-standard questions) are often more practical, even if theoretically less principled.

5.12 Summary

- When **multiple stakeholders** have conflicting preferences, aggregation is unavoidable—and impossibility results constrain what any aggregation rule can achieve.
- **Arrow's Impossibility Theorem** shows that no rule for three or more alternatives can simultaneously satisfy Unanimity, IIA, and Non-dictatorship. The **Gibbard-Satterthwaite Theorem** adds that non-dictatorial rules are always manipulable.
- **Domain restrictions** provide an escape: when preferences are **single-peaked** (each voter has a unique ideal point on an ordered spectrum), the **generalized median voter scheme** is both strategy-proof and Pareto efficient (Moulin's theorem).
- The **DPO-Borda connection** is a key bridge between this chapter and modern ML: DPO implicitly optimizes the expected Borda score when aggregating across annotators, linking social choice theory directly to language model alignment.
- **Nosy vs. non-nosy preferences** (Sen's framework) distinguish personal choices from preferences about others' behavior. The **liberal paradox** shows that respecting individual rights can conflict with Pareto efficiency—a tension that AI systems must navigate explicitly.
- The **inversion problem**—that observed behavior does not necessarily reflect underlying preferences—undermines revealed preference approaches and has direct implications for RLHF, where annotator fatigue, framing, and incentives distort the feedback signal.
- **Mechanism design without money** (peer prediction, the Mutual Information Paradigm) provides tools for eliciting truthful reports when standard payment-based incentives are unavailable, as in crowdsourced annotation.
- **Community Notes** exemplifies real-world preference aggregation: its factor model separates note quality from ideological slant, surfacing content that earns cross-partisan agreement.

5.13 Quick Check

Test your understanding before proceeding to the exercises.

1. Arrow's theorem requires three axioms. Which axiom does the Borda count violate, and how?

2. Under single-peaked preferences with 5 voters whose peaks are $\{2, 4, 5, 7, 9\}$, what is the pure median outcome? Can any voter manipulate this outcome by misreporting their peak?
3. In the Community Notes model, why does controlling for the $p_i \cdot q_j$ interaction term improve note quality estimation compared to simple averaging?
4. Give an example of a "nosy preference" in AI content moderation. Why does Sen's liberal paradox make it impossible to both respect individual rights and achieve Pareto efficiency when nosy preferences are present?

5.14 Discussion Questions

1. **Arrow's theorem and cardinal utilities**: Arrow's Impossibility Theorem applies to ordinal preferences. How does allowing cardinal utilities (like summing utilities across voters) change the picture? What new problems arise when we allow interpersonal utility comparisons?
2. **Single-peaked preferences in practice**: Can you think of real-world preference domains that are plausibly single-peaked? What about domains that clearly violate single-peakedness? How might you test whether a given preference dataset is approximately single-peaked?
3. **DPO and Borda**: The DPO-Borda connection suggests that DPO finds the response that wins the most pairwise comparisons. Is this what we actually want for language model alignment? What are scenarios where the Borda winner might not be the "best" response?
4. **Nosy preferences in content moderation**: Give three examples of nosy preferences that arise in content moderation for social media platforms. For each, discuss whether you think the platform should respect or override this preference.
5. **The inversion problem**: How should RLHF systems handle the possibility that annotator choices don't reflect their true preferences (due to fatigue, cognitive biases, or the structure of the annotation task)? Propose a concrete intervention.
6. **Privacy vs. personalization**: A streaming service wants to personalize recommendations but users are concerned about privacy. Using the Contextual Integrity framework, design a data policy that balances these concerns. What transmission principles would you specify?
7. **Paternalism trade-offs**: Consider an AI assistant that notices a user frequently asks questions late at night that they seem to regret in the morning. Should the assistant intervene? If so, how? Discuss the ethical considerations.
8. **Mechanism design without money**: In RLHF, we cannot pay annotators based on the "truth" of their labels (since there's no ground truth). How do peer prediction mechanisms like MIP help? What assumptions do they require?

9. **Community Notes vs. majority voting**: Why does Community Notes use a factor model instead of simple majority voting? What problems would majority voting create for controversial topics?

10. **Strategic annotators**: If RLHF annotators knew their labels would influence the model's behavior, would they have incentives to vote strategically? How does Gibbard–Satterthwaite apply here?

11. **Auction design for AI compute**: Cloud providers sell AI compute resources. Design an auction mechanism for allocating GPU time that maximizes welfare. What properties should it have?

12. **VCG and AI alignment**: Could VCG-like mechanisms be used to aggregate diverse stakeholder preferences for AI system design? What would the "payments" be in this context?

13. **Pluralistic alignment**: Instead of aggregating preferences into a single objective, some propose that AI systems should preserve preference diversity. How would you operationalize this? What are the trade-offs?

14. **Separability across criteria**: When is it reasonable to assume that preferences over "helpful," "harmless," and "honest" are separable? When does this assumption fail? Give concrete examples.

15. **Contextual Integrity violations**: Identify three ways that current AI systems might violate contextual integrity norms. For each, propose a design change that would restore contextual integrity.

5.15 Bibliographic Notes

Social Choice Theory: The foundational results in social choice are due to Arrow (1951), who proved his impossibility theorem, and Gibbard (1973) and Satterthwaite (1975), who independently proved the manipulability theorem. Black (1948) introduced single-peaked preferences, and Moulin (1980) characterized strategy-proof rules on this domain. For comprehensive treatments of social choice theory, see Sen (1970a) and Austen-Smith and Banks (2000).

Domain Restrictions: Beyond single-peakedness, various restricted domains have been studied. See Gaertner (2001) for a survey. The connection between single-peaked preferences and the median voter theorem goes back to Black (1948) and Downs (1957).

Borda Count and Scoring Rules: The Borda count was proposed by Jean-Charles de Borda in 1781. Modern treatments include Young (1974), who characterized Borda axiomatically. The connection between Borda and pairwise comparisons is explored in Fishburn (1977). The DPO-Borda connection builds on insights from Rafailov et al. (2023).

Nosy Preferences and Liberalism: Sen's (1970b) Liberal Paradox sparked extensive literature on the conflict between liberty and efficiency. See Hausman (2012) for philosophical discussion and Gaertner and Heinecke (1988) for domain restriction approaches.

Contextual Integrity: Nissenbaum's (2009) framework has been influential in privacy scholarship. Applications to machine learning appear in recent work on contextual data use.

Mechanism Design: Vickrey (1961) introduced the second-price auction. The VCG mechanism was developed by Vickrey, Clarke (1971), and Groves (1973). Myerson's (1981) optimal auction theory is foundational. The Bulow-Klemperer theorem appears in Bulow and Klemperer (1996). For textbook treatments, see Krishna (2009) and Börgers (2015).

Peer Prediction: The Mutual Information Paradigm was introduced by Kong and Schoenebeck (2019). Earlier work on peer prediction includes Miller, Resnick, and Zeckhauser (2005) and Prelec (2004). Applications to peer grading appear in Hartline et al. (2020) and Hartline et al. (2023).

Incentive-Compatible Bandits: The study of exploration under strategic agents is developed in Mansour, Slivkins, and Syrgkanis (2019) and Mansour et al. (2021). Connections to information design are explored in Bergemann and Morris (2019).

AI Alignment and Aggregation: Jury learning and methods for aggregating diverse annotator preferences are studied in Gordon et al. (2022). Scalable oversight through recursive reward modeling appears in Christiano, Shlegeris, and Amodei (2018). AI-assisted debate is proposed in Irving, Christiano, and Amodei (2018). Community Notes is described in X/Twitter's public documentation.

Essential Reading

If you read only five papers from this chapter's references, make them these:

1. **Arrow (1951)** — Arrow's Impossibility Theorem: the foundational impossibility result for preference aggregation
2. **Gibbard (1973)** — The Gibbard-Satterthwaite theorem: strategy-proofness is incompatible with non-dictatorship
3. **Sen (1970b)** — The Impossibility of a Paretian Liberal: liberty vs. efficiency in social choice
4. **Gordon et al. (2022)** — Jury learning: aggregating diverse annotator preferences for fairer AI systems
5. **Kong and Schoenebeck (2019)** — The Mutual Information Paradigm for incentive-compatible preference elicitation

5.16 Further Reading

For readers who want to go deeper into the topics introduced in this chapter, we recommend the following:

- **Sen (1970a)**, *Collective Choice and Social Welfare* — Sen's classic text connecting social choice theory to welfare economics, providing the rigorous foundations for Arrow's theorem, the liberal paradox, and interpersonal comparability of utility.
- **Börgers (2015)**, *An Introduction to the Theory of Mechanism Design* — An accessible introduction to auctions, VCG mechanisms, and incentive compatibility, covering the game-theoretic foundations of the mechanism design topics in this chapter.

- **Austen-Smith and Banks (2000)**, *Positive Political Theory I* — A rigorous treatment of voting theory, impossibility results, and domain restrictions, suitable for readers who want the formal proofs behind the results surveyed here.
- **List and Pettit (2011)**, *Group Agency* — Extends impossibility results from preference aggregation to judgment aggregation (aggregating beliefs, not just preferences), showing that related impossibilities arise when groups try to maintain consistent collective beliefs.
- **Gordon et al. (2022)**, "Jury Learning: Integrating Dissenting Voices into Machine Learning" — Proposes methods for integrating diverse annotator perspectives rather than collapsing to majority vote, directly relevant to pluralistic approaches to LLM alignment.
- **Miller, Resnick, and Zeckhauser (2005)**, "Eliciting Informative Feedback: The Peer-Prediction Method" — The foundational work on eliciting truthful reports without verification, providing the theoretical basis for the peer prediction and MIP frameworks discussed in this chapter.
- **Conitzer et al. (2024)**, "Social Choice for AI Policy" — Recent work on applying social choice theory to AI governance and alignment, arguing for maintaining multiple preference models rather than collapsing to a single aggregate.

5.17 Exercises

5.17.1 Exercise 1: Condorcet Cycles (*)

Consider three voters with the following preferences over alternatives $\{A, B, C\}$:

- Voter 1: $A \succ B \succ C$
- Voter 2: $B \succ C \succ A$
- Voter 3: $C \succ A \succ B$

(a) Show that majority rule produces a preference cycle. Specifically, determine the majority preference for each pair (A, B), (B, C), and (A, C).

(b) Compute the Borda score for each alternative. Which alternative wins under Borda count?

(c) Does this preference profile have a Condorcet winner? Explain.

5.17.2 Exercise 2: Single-Peaked Preferences (**)

Consider five voters with peaks at positions $\{1, 3, 5, 7, 9\}$ on a line (total order).

(a) What is the outcome of the pure median rule (no phantom voters)?

(b) Add two phantom voters at positions 2 and 8. Compute the generalized median outcome.

(c) Suppose voter 1 (with true peak at 1) misreports their peak as 0. Does this change the outcome? Use this to argue informally why the generalized median is strategy-proof.

5.17.3 Exercise 3: DPO-Borda Connection (**)

Consider three responses $\{y_1, y_2, y_3\}$ to a prompt x. Suppose annotators have the following pairwise preferences:

- $\mathbb{P}(y_1 \succ y_2) = 0.7$
- $\mathbb{P}(y_1 \succ y_3) = 0.6$
- $\mathbb{P}(y_2 \succ y_3) = 0.8$

(a) Compute the expected Borda score for each response (sum of pairwise win probabilities).

(b) If responses are sampled uniformly at random from a reference policy, which response would DPO upweight most strongly? Connect to the DPO-Borda theorem.

5.17.4 Exercise 4: Virtual Valuations and Optimal Reserve (**)

For bidders with values drawn i.i.d. from Uniform$[0, 1]$:

(a) Derive the virtual valuation function $\varphi(v) = v - \frac{1-F(v)}{f(v)}$.

(b) Compute the optimal reserve price r^* by solving $\varphi(r^*) = 0$.

(c) With two bidders, compute the expected revenue of a second-price auction with reserve $r = 0.5$ and compare to expected revenue without a reserve.

5.17.5 Exercise 5: VCG Payments (**)

In a combinatorial auction with items $\{A, B\}$ and three bidders with the following valuations:

Bidder	$v(\{A\})$	$v(\{B\})$	$v(\{A, B\})$
1	3	2	4
2	2	4	5
3	1	1	6

(a) Find the welfare-maximizing allocation.

(b) Compute the VCG payment for each winner using the externality formula.

5.17.6 Exercise 6: Nosy Preferences (***)

Formalize the Paretian Liberal paradox for a content moderation setting:

(a) Define the set of "content types" $C = \{c_1, c_2, c_3\}$ and two users with nosy preferences over what content each other should see.

(b) Construct a preference profile where minimal liberalism (each user decides their own feed) conflicts with Pareto efficiency.

(c) Propose a domain restriction that would resolve the paradox in this setting.

5.17.7 Exercise 7: Inversion Problem in RLHF (**)

An RLHF annotator's preferences become more random as they fatigue. Model this as follows: in the first hour, preferences follow Bradley-Terry with true utilities. After the first hour, choices become uniform random with probability $p_{\text{fatigue}} = 0.3$.

(a) Write the likelihood of observing $y_w \succ y_l$ as a mixture model.

(b) Suppose you observe inconsistent annotations from an annotator (e.g., $y_1 \succ y_2$ and later $y_2 \succ y_1$). How would you detect whether this is due to fatigue vs. genuinely indifferent preferences?

(c) Propose a weighting scheme to down-weight fatigued annotations in reward model training.

5.17.8 Exercise 8: Community Notes Factor Model (**)

Implement a simplified Community Notes model. Generate synthetic data as follows:

- 100 users with ideological positions $p_i \sim \text{Uniform}[-1, 1]$
- 20 notes with true quality $\beta_j \sim \mathcal{N}(0, 1)$ and ideological slant $q_j \sim \text{Uniform}[-1, 1]$
- Each user-note rating: $r_{ij} = \beta_j + p_i \cdot q_j + \varepsilon_{ij}$ where $\varepsilon_{ij} \sim \mathcal{N}(0, 0.5)$

(a) Implement code to generate this synthetic data.

(b) Use linear regression to estimate β_j (note quality) controlling for the ideological interaction term $p_i \cdot q_j$.

(c) Compare your estimated $\hat{\beta}_j$ to the true β_j. How well does the model recover note quality?

5.17.9 Exercise 9: Pairwise Feedback Mechanisms for Digital Goods (***)

Consider a marketplace for digital goods (such as personalized articles, artwork, or AI-generated data), where the exact utility derived from these goods is only revealed to buyers after the goods have been generated and delivered. To elicit truthful preferences from buyers who find it difficult to precisely quantify their valuations beforehand, the marketplace implements a pairwise feedback mechanism, inspired by the work of Robertson and Koyejo (2023).

Formally, each buyer requests a personalized digital good and, upon receiving the good, provides feedback by indicating whether their realized utility is higher or lower than a randomly chosen reference price $c \in [0, 1]$. The mechanism utilizes this binary feedback to estimate valuations and allocate future goods accordingly.

Answer the following:

(a) **Formalize the Problem**: Let u_i denote the true valuation of buyer i, and let $r_i(c)$ denote the buyer's reported feedback ($r_i(c) = 1$ if $u_i \geq c$, 0 otherwise). Prove that, under uniform random selection of the reference price c, the expected value $\mathbb{E}[r_i(c)]$ is equal to the true valuation u_i.

(b) **Incentive Compatibility Analysis**: Discuss conditions under which this feedback-based mechanism is incentive compatible, i.e., buyers have no incentive to misreport their preferences. Specifically, analyze why strategic misreporting (reporting $r_i(c)$ incorrectly for some reference prices) would not increase a buyer's expected payoff.

(c) **Regret Analysis**: Suppose the mechanism estimates buyers' utilities from past feedback and allocates future goods using an epsilon-greedy strategy (exploration rate η_t). Provide an informal discussion of the trade-off involved in choosing the exploration rate, and how it affects the social welfare and revenue of the marketplace over time.

(d) **Practical Implications**: Suggest one practical scenario outside the digital-goods marketplace where such a feedback-driven, pairwise comparison approach would be beneficial. Briefly justify your choice, mentioning challenges and benefits.

5.17.10 Exercise 10: Scalable Oversight in Complex Decision-Making (***)

In scenarios involving complex or high-dimensional outcomes (such as summarizing lengthy texts, assessing the quality of detailed AI-generated reports, or reviewing scientific papers), evaluating the quality of outputs can become infeasible for a single human overseer. One practical solution is scalable oversight, where the evaluation task is decomposed and distributed among multiple human evaluators or even assisted by AI agents. Consider a scalable oversight scenario inspired by recursive reward modeling, where complex evaluations are hierarchically decomposed into simpler tasks. Specifically, suppose you want to evaluate a lengthy report generated by an AI system. Answer the following:

(a) Decomposition of the Task: Propose a formal recursive decomposition strategy to evaluate a long AI-generated report of length (N) paragraphs. Specifically, describe a hierarchical evaluation method that decomposes the original evaluation into simpler subtasks at multiple hierarchical levels. Clearly describe how many subtasks you have at each level and how the final aggregated evaluation is computed.

(b) Statistical Aggregation Method: Suppose each evaluation subtask yields a binary score (s_i □{0,1}), where (1) indicates acceptable quality and (0) indicates unacceptable quality. Propose a simple statistical aggregation method (e.g., majority voting, threshold voting, weighted aggregation, etc.) to combine subtask evaluations into a single global quality assessment at the top level. Justify your choice mathematically.

(c) Computational Simulation: Implement a Python simulation of your hierarchical decomposition and aggregation method described in parts (a) and (b). Assume each subtask is evaluated with some fixed probability (p) of being correct (representing human evaluators with bounded accuracy).

Specifically, your simulation should: - Implement a hierarchical evaluation scheme (e.g., binary-tree decomposition). - Assume evaluators have accuracy (p = 0.8) (i.e., probability of correctly identifying paragraph quality). - Simulate how evaluator accuracy at the leaf nodes affects the reliability of the global evaluation at the root node. - Plot how the reliability of the top-level evaluation (accuracy at the root) varies as you increase the depth of hierarchy for a report of fixed length (e.g., (N = 64) paragraphs).

(d) Practical Discussion: Briefly discuss advantages and potential drawbacks of scalable oversight approaches such as recursive decomposition in the context of AI alignment. Include considerations such as evaluator fatigue, consistency, cost, and vulnerability to manipulation or collusion.

References

Arrow, Kenneth J. 1951. *Social Choice and Individual Values*. John Wiley; Sons.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. "Finite-Time Analysis of the Multiarmed Bandit Problem." *Machine Learning* 47 (2). https://doi.org/10.1023/A:1013689704352.

Austen-Smith, David, and Jeffrey S Banks. 2000. *Positive Political Theory i: Collective Preference*. University of Michigan Press.

Bergemann, Dirk, and Stephen Morris. 2019. "Information Design: A Unified Perspective." In *Journal of Economic Literature*, 57:44–95. 1.

Black, Duncan. 1948. "On the Rationale of Group Decision-Making." *Journal of Political Economy* 56 (1): 23–34.

Börgers, Tilman. 2015. *An Introduction to the Theory of Mechanism Design*. Oxford University Press.

Bulow, Jeremy, and Paul Klemperer. 1996. "Auctions Versus Negotiations." *The American Economic Review* 86 (1): 180–94. http://www.jstor.org/stable/2118262.

Christiano, Paul, Buck Shlegeris, and Dario Amodei. 2018. "Supervising Strong Learners by Amplifying Weak Experts." *arXiv Preprint arXiv:1810.08575*.

Clarke, Edward H. 1971. "Multipart Pricing of Public Goods." *Public Choice* 11: 17–33.

Conitzer, Vincent, Walter Sinnott-Armstrong, Jana Schaich Borg, Finale Doshi-Velez, and Maria Gini. 2024. "Social Choice for AI Policy: Lessons from Recent Work." *arXiv Preprint arXiv:2404.16412*.

Downs, Anthony. 1957. *An Economic Theory of Democracy*. Harper; Row.
Fishburn, Peter C. 1977. "Condorcet Social Choice Functions." *SIAM Journal on Applied Mathematics* 33 (3): 469–89.
Gaertner, Wulf. 2001. *Domain Conditions in Social Choice Theory*. Cambridge University Press.
Gaertner, Wulf, and Achim Heinecke. 1988. "When Does the Condorcet Winner Exist? A Geometrical Approach." *Social Choice and Welfare* 5: 341–50.
Gibbard, Allan. 1973. "Manipulation of Voting Schemes: A General Result." *Econometrica* 41 (4): 587–601.
Gordon, Noah J., Vaishnavh Nagarajan Shankar, Shi Feng, Yejin Choi, and Noah A. Smith. 2022. "Jury Learning: Integrating Dissenting Voices into Machine Learning Models." In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2658–73. Association for Computational Linguistics.
Groves, Theodore. 1973. "Incentives in Teams." *Econometrica* 41 (4): 617–31.
Hartline, Jason D., Yingkai Li, Liren Shan, and Yifan Wu. 2020. "Optimization of Scoring Rules." *CoRR* abs/2007.02905. https://arxiv.org/abs/2007.02905.
Hartline, Jason D., Liren Shan, Yingkai Li, and Yifan Wu. 2023. "Optimal Scoring Rules for Multi-Dimensional Effort." In *Proceedings of Thirty Sixth Conference on Learning Theory*, edited by Gergely Neu and Lorenzo Rosasco, 195:2624–50. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v195/hartline23a.html.
Hausman, Daniel M. 2012. *Preference, Value, Choice, and Welfare*. Cambridge University Press.
Irving, Geoffrey, Paul Christiano, and Dario Amodei. 2018. "AI Safety via Debate." *arXiv Preprint arXiv:1805.00899*.
Kong, Yuqing, and Grant Schoenebeck. 2019. "An Information Theoretic Framework for Designing Information Elicitation Mechanisms That Reward Truth-Telling." *ACM Trans. Econ. Comput.* 7 (1). https://doi.org/10.1145/3296670.
Krishna, Vijay. 2009. *Auction Theory*. 2nd ed. Academic Press.
Lai, T. L, and Herbert Robbins. 1985. "Asymptotically Efficient Adaptive Allocation Rules." *Advances in Applied Mathematics* 6 (1): 4–22. https://doi.org/https://doi.org/10.1016/0196-8858(85)90002-8.
List, Christian, and Philip Pettit. 2011. *Group Agency: The Possibility, Design, and Status of Corporate Agents*. Oxford: Oxford University Press.
Mansour, Yishay, Aleksandrs Slivkins, and Vasilis Syrgkanis. 2019. "Bayesian Incentive-Compatible Bandit Exploration." https://arxiv.org/abs/1502.04147.
Mansour, Yishay, Aleksandrs Slivkins, Vasilis Syrgkanis, and Zhiwei Steven Wu. 2021. "Bayesian Exploration: Incentivizing Exploration in Bayesian Games." https://arxiv.org/abs/1602.07570.
Miller, Nolan, Paul Resnick, and Richard Zeckhauser. 2005. "Eliciting Informative Feedback: The Peer-Prediction Method." In *Management Science*, 51:1359–73. 9.
Moulin, Hervé. 1980. "On Strategy-Proofness and Single Peakedness." *Public Choice* 35 (4): 437–55.
Myerson, Roger B. 1981. "Optimal Auction Design." *Mathematics of Operations Research* 6 (1): 58–73.
Nissenbaum, Helen. 2009. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press.
Prelec, Dražen. 2004. "A Bayesian Truth Serum for Subjective Data." In *Science*, 306:462–66. 5695.
Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." https://arxiv.org/abs/2305.18290.
Russo, Daniel, and Benjamin Van Roy. 2015. "An Information-Theoretic Analysis of Thompson Sampling." https://arxiv.org/abs/1403.5341.
Satterthwaite, Mark Allen. 1975. "Strategy-Proofness and Arrow's Conditions: Existence and Correspondence Theorems for Voting Procedures and Social Welfare Functions." *Journal of Economic Theory* 10 (2): 187–217.
Sen, Amartya. 1970a. *Collective Choice and Social Welfare*. Holden-Day.
———. 1970b. "The Impossibility of a Paretian Liberal." *Journal of Political Economy* 78 (1): 152–57. https://doi.org/10.1086/259614.

Vickrey, William. 1961. "Counterspeculation, Auctions, and Competitive Sealed Tenders." *Journal of Finance* 16 (1): 8–37.
Young, H Peyton. 1974. "An Axiomatization of Borda's Rule." *Journal of Economic Theory* 9 (1): 43–52.

6 Whose?

Note

By the end of this chapter you will be able to:

- **Identify** how technical design choices at each stage of the preference learning pipeline (elicitation, learning, aggregation, decision) embed value judgments about whose preferences matter and how they should be weighted.
- **Explain** the inversion problem—why behavior mental state—and how optimizing for revealed preferences (clicks, bids, engagement) can systematically disadvantage groups whose behavior diverges from preferences due to context (fatigue, time constraints, cultural norms).
- **Distinguish** between individual fairness (similar entities treated similarly) and group fairness (protected groups have equal outcomes), recognizing their fundamental incompatibility as demonstrated by Dwork et al.
- **Apply** Sen's framework of nosy vs. non-nosy preferences to determine when liberal assistance (respecting stated preferences) is insufficient and illiberal assistance (overriding preferences) is justified.
- **Analyze** how unfairness compounds across the elicitation → learning → aggregation → decision pipeline through feedback loops, showing how small biases at each stage multiply over time.
- **Evaluate** governance mechanisms (designer discretion, user control, participatory design, regulatory oversight, professional standards) for deciding whose preferences should be weighted and how.
- **Design** preference learning systems that make value tradeoffs explicit, document who benefits and who is harmed, and include fairness constraints at each pipeline stage.
- **Implement** stratified sampling, subgroup performance monitoring, and fairness-constrained optimization to detect and mitigate compounding unfairness.
- **Audit** systems for red flags: feedback loops where average performance improves but subgroup performance worsens, optimization-fairness divergence, invisibility of harm, post-hoc rationalization, ignoring context, and revealed preference fallacies.
- **Connect** abstract fairness concepts to concrete technical decisions in Bradley-Terry models (Chapter 2), DPO weighting (Chapter 3), active learning via Fisher information (Chapter 4), exploration strategies (Chapter 5), and aggregation rules (Chapter 6).

Suggested Lecture Plan

This chapter can be covered in two 50-minute lectures plus a discussion/activity session:

Lecture 1 (Sections 7.1–7.2): The Pipeline and Design Choices

- Introduction: Whose preferences matter? The central question (10 min)
- The 4-stage pipeline framework with AI Review Assistant running example (10 min)
- Elicitation: Who gets queried? The inversion problem (10 min)

– Learning: What structures are valid? IIA violations and context-dependence (10 min)
– Aggregation and decision: Weighing preferences, liberal vs. illiberal assistance (10 min)

Lecture 2 (Sections 7.3–7.6): Fairness and Design Principles

– Individual vs. group fairness: formal definitions, conflicts, examples (15 min)
– Process vs. outcome fairness: tensions and connections to Sen's liberalism (10 min)
– How unfairness compounds: feedback loops and end-to-end analysis (10 min)
– Eight design principles for practitioners (10 min)
– Red flags, governance mechanisms (5 min)

Discussion/Activity Session: Case Study Analysis (50 min)

– Paper-reviewer matching design exercise: make choices at each pipeline stage (30 min)
– Discussion of case studies: LaMPost, Multi-Value, DaDa (20 min)

6.1 Chapter Overview

Chapters 2–6 provided powerful technical methods for learning from human preferences:

– **Chapter 2** showed how to model preferences through Bradley-Terry, Rasch, and factor models, assuming preferences exist and can be represented mathematically.
– **Chapter 3** developed estimation methods (MLE, Bayesian inference, online learning), assuming we have preference data to train on.
– **Chapter 4** introduced active elicitation via Fisher information and optimal design, assuming we know which comparisons to request.
– **Chapter 5** addressed sequential decisions under preference uncertainty, assuming we know what outcomes to pursue.
– **Chapter 6** covered aggregation mechanisms, providing technical tools but not normative guidance on whose preferences to aggregate.

This chapter provides the missing normative framework: **whose preferences should we learn from, and how should we weigh them?**

Every technical choice embeds value judgments. Choosing an active learning strategy determines *who gets asked* (efficiency vs. representation). Assuming IIA imposes structure on *what preferences are valid* (rational preferences vs. psychological reality). Selecting DPO's reference policy π_{ref} determines *whose preferences count more* (high-volume users vs. equal weighting). Deciding when to override stated preferences determines *what preferences are respected* (liberal vs. illiberal assistance).

The Central Insight: These are not merely technical decisions—they are fairness decisions with real consequences. A system meant to "help all reviewers" may primarily help already-privileged users if we optimize for efficiency at each stage. Small biases compound: elicitation undersamples some groups → learning fits them poorly → aggregation downweights them → decisions provide poor assistance → they disengage → even less data. The gap widens.

The Roadmap: This chapter traces the preference learning pipeline from elicitation through decision-making, analyzing how each stage embeds values and how unfairness accumulates. We formalize key fairness concepts (individual vs. group, process vs. outcome), provide concrete design principles for building fairer systems, examine governance mechanisms for value decisions, and ground the analysis in real-world case studies.

Connection to Earlier Chapters:

- IIA from Bradley-Terry (Chapter 2,) has fairness implications: it privileges "rational" preferences and may mismodel overburdened users.
- DPO's reference policy (Chapter 3) implicitly weights by participation—high-volume users dominate if we don't correct for unequal engagement.
- Active learning via Fisher information (Chapter 4) maximizes information gain but may undersample minority groups, creating representativeness issues.
- Exploration strategies (Chapter 5) raise liberal vs. illiberal questions: explore what users *want* or what's *best* for them?
- Aggregation impossibilities (Chapter 6, Arrow's theorem, Sen's Paretian Liberal) show that no mechanism satisfies all properties—fairness constraints further limit options.

This chapter equips you to recognize value tradeoffs, make conscious fairness choices, and build systems that respect human dignity while acknowledging impossibility results and practical constraints.

6.2 Introduction: Whose Preferences Matter?

This book has taught you *how* to learn from human preferences. You can model preferences through Bradley-Terry (), estimate parameters via maximum likelihood (), actively collect informative comparisons (), make sequential decisions (), and aggregate heterogeneous preferences (). These are powerful technical tools.

But now we ask a deeper question: **Whose preferences should we learn from, and how should we weigh them?**

This is not a technical question with an algorithmic answer—it is a *normative* question about values and fairness. Yet every technical decision you make embeds an answer to this question, whether you recognize it or not:

- When you choose an active learning strategy (Chapter 4), you determine **who gets asked**. Maximizing Fisher information is efficient but may undersample minority users. This is a fairness decision disguised as an optimization problem.
- When you assume IIA for tractability (Chapter 2), you impose structure on **what preferences are valid**. Context-dependent preferences (fatigue, framing, load) are treated as "irrational" and ignored. But these violations disproportionately affect overburdened users.

- When you use DPO with a reference policy π_{ref} (Chapter 3), you determine **whose preferences count more**. High-volume users dominate the reference policy. Is this fair? Or should all users be weighted equally?
- When you decide whether to defer to stated preferences or override them (Chapter 5), you determine **what preferences are respected**. Should an AI review assistant help a reviewer be consistently harsh (liberal assistance) or nudge toward constructive feedback (illiberal assistance)?

These choices have consequences. They determine who benefits from your system and who is harmed.

6.2.1 The Four-Stage Pipeline

To analyze these fairness questions systematically, we introduce a **four-stage pipeline** for preference learning systems:

1. **Elicitation** ($\mathcal{E}$): How do we collect preference data? Who do we query? What questions do we ask?
2. **Learning** ($\mathcal{L}$): What model structure do we assume? What preference patterns are considered valid?
3. **Aggregation** ($\mathcal{A}$): How do we combine multiple users' preferences? Whose preferences are weighted more?
4. **Decision** ($\mathcal{D}$): When do we defer to user preferences vs. override them? What constitutes legitimate paternalism?

At each stage, we make choices that embed values. Often these choices compound: a bias introduced at elicitation amplifies through learning, aggregation, and decision-making, creating feedback loops that widen disparities over time.

> **! IDENTIFYING STAKEHOLDERS SYSTEMATICALLY**
>
> Before analyzing fairness at each pipeline stage, one must first identify **who is affected**. A useful framework distinguishes four categories: (1) **direct users** who interact with the system and provide preference data (e.g., reviewers using the AI assistant); (2) **affected parties** who are impacted by decisions but may not be users (e.g., paper authors whose work is reviewed); (3) **system designers** who set objectives and constraints (e.g., conference organizers); and (4) **society** broadly, which bears indirect consequences (e.g., the research community's trust in peer review). Different stakeholder groups may have conflicting interests, and a fairness analysis that considers only direct users can overlook harms to affected parties—a common failure mode in deployed preference learning systems.

THE CENTRAL THESIS

Every technical choice in the preference learning pipeline is a value choice about whose preferences matter and how they should be weighted.

You cannot avoid making these choices. You can only choose whether to make them *consciously* and *transparently*, or to hide them behind claims of "technical neutrality."

This chapter provides frameworks for recognizing these choices, analyzing their fairness implications, and making principled tradeoffs.

COMMON PITFALL: ASSUMING A SINGLE 'FAIR' SOLUTION EXISTS

A natural instinct is to treat fairness as an optimization problem — "find the system that is most fair." But the impossibility results throughout this book (Arrow's theorem, Gibbard-Satterthwaite, the incompatibility of individual and group fairness) mean there is **no single solution** that satisfies all fairness desiderata simultaneously. A system that is fair by one metric (e.g., calibration across groups) may be unfair by another (e.g., equalized odds). The pitfall is claiming technical neutrality: every design choice — what data to collect, which loss to minimize, how to aggregate, when to override — embeds a value judgment about whose preferences matter. The goal is not to find *the* fair solution but to make these tradeoffs **consciously and transparently**.

6.2.2 Running Example: AI Review Assistant for Peer Review

To ground these abstract questions, we use a concrete running example throughout this chapter: an AI system that helps peer reviewers write paper reviews. The system learns from each reviewer's past reviews to assist them in evaluating new papers.

The Setting: Consider a large ML conference with 10,000 submitted papers and 5,000 reviewers. Each paper needs at least 3 reviews. Reviewers bid on papers (prefer / willing / not-willing), and a matching algorithm assigns papers to reviewers. An AI assistant learns from past reviews to help reviewers write better, more efficient reviews.

Why This Example? Peer review combines all the challenges of preference learning:

- **Elicitation**: Which reviewers should we query for more feedback to train the assistant? Senior reviewers write more detailed reviews (more training data), but this may leave junior reviewers under-served.
- **Learning**: Reviewers have context-dependent preferences (fatigue, load, framing). Should we assume IIA? Or model context explicitly?
- **Aggregation**: Should the shared model weight all reviewers equally? Or weight by review volume (more data from seniors)?
- **Decision**: If a reviewer tends to write harsh reviews, should the assistant help them be consistently harsh? Or nudge toward constructive feedback?

Moreover, peer review has clear stakeholders (reviewers, authors, editors, scientific community), measurable outcomes (review quality, author satisfaction, acceptance rates), and real fairness concerns (junior vs. senior reviewers, underrepresented subfields, institutional prestige).

We will trace how design choices at each pipeline stage create fairness implications, and how these compound into systematic disadvantages for some groups.

6.2.3 Connection to Earlier Chapters

This chapter builds directly on the technical foundations from Chapters 2–6:

From Chapter 2 (Foundations): The Bradley-Terry model assumes IIA—if reviewer prefers Paper A over Paper B, adding Paper C shouldn't change this. But in , we'll see why this assumption has fairness consequences: it treats context-dependent preferences as invalid, disadvantaging overburdened reviewers.

From Chapter 3 (Learning): DPO maximizes a weighted Borda rule where π_{ref} is trained on existing data (). But existing data reflects existing inequities. In , we'll analyze how status quo bias in π_{ref} weights senior reviewers more heavily.

From Chapter 4 (Elicitation): Active learning via Fisher information () queries where the model is most uncertain. But uncertainty may be highest for well-represented groups with complex preferences. In , we'll show how information-maximizing strategies can undersample minorities.

From Chapter 5 (Decisions): Thompson Sampling explores based on posterior uncertainty (). But should we explore over what users *want* (liberal) or what's *best* for them (illiberal)? In , we'll formalize this distinction using Sen's framework of nosy preferences.

From Chapter 6 (Aggregation): Arrow's theorem and Sen's Impossibility of a Paretian Liberal () show that no aggregation mechanism satisfies all desirable properties. In , we'll see how adding fairness constraints further limits our options.

The technical methods are powerful. But applying them responsibly requires understanding *whose* preferences you're optimizing for and *whose* interests you're serving.

6.2.4 Roadmap for This Chapter

The rest of this chapter proceeds as follows:

- analyzes the four-stage pipeline in detail, showing how each stage embeds value judgments and how unfairness compounds across stages.
- formalizes key fairness distinctions: individual vs. group fairness, process vs. outcome fairness, and the fundamental incompatibilities between them.

- provides eight concrete design principles for building fairer preference learning systems, along with red flags to watch for.
- **Governance** (discussed throughout the chapter) examines governance mechanisms for deciding whose preferences matter: designer discretion, user control, participatory design, regulatory oversight, and professional standards.
- **Design Principles** () distills the chapter's insights into actionable principles for building fairer preference learning systems.

Let's begin by tracing the AI review assistant through the four-stage pipeline, seeing how fairness issues emerge and compound at each step.

6.3 *Design Decisions as Value Choices*

WORKED EXAMPLE: VALUE CHOICES IN THE AI REVIEW ASSISTANT PIPELINE

Consider building an AI assistant for peer review. At each pipeline stage, a seemingly neutral technical choice embeds a value judgment.

Stage	Technical Choice	Value Embedded	Who Benefits	Who Is Harmed
Elicitation $\mathcal{E}$	Query reviewers with highest Fisher information	Efficiency over representation	Well-represented subfields	Niche/emerging areas
Learning $\mathcal{L}$	Fit Bradley-Terry (assumes IIA)	Assumes context-independent preferences	Average-context reviewers	Overburdened reviewers whose preferences are context-dependent
Aggregation $\mathcal{A}$	Simple majority across annotators	Equal weight per person	Majority subfield	Interdisciplinary or minority perspectives
Decision $\mathcal{D}$	Always follow the model's recommendation	Trust the system	Users well-served by the model	Edge cases, novel paper types

Compounding effect: If elicitation undersamples postdocs → the model learns senior-faculty preferences → aggregation weights those preferences equally (but they're overrepresented in data) → the assistant serves senior faculty well, who use it more, generating more data. A 10% sampling bias at stage $\mathcal{E}$ can become a 40% performance gap after one feedback cycle.

Intervention: Stratified sampling at $\mathcal{E}$ (ensure equal representation by career stage), fairness-constrained learning at $\mathcal{L}$ (equalize subgroup AUC), and regular auditing at $\mathcal{D}$ (compare subgroup satisfaction) break the feedback loop.

We now trace the AI review assistant example through all four pipeline stages, analyzing the value tradeoffs and fairness implications at each step.

6.3.1 *Elicitation: Who Gets Queried?*

Technical Question: How should we collect data to train the review assistant? Which reviewers should we observe more carefully?

Consider three design options for elicitation policy $\mathcal{E}$:

1. **Universal querying**: Observe all reviewers equally (every review provides training data)
2. **Productivity-based**: Query "productive" reviewers more—those who write detailed, timely reviews
3. **Active learning**: Use Fisher information () to query where the model is most uncertain

Value Embedded: Options 2 and 3 prioritize *efficiency* (maximize information gain per query) over *representation* (ensure all reviewer types contribute training data).

This seems like a purely technical optimization decision. But it has profound fairness consequences.

6.3.1.1 *The Inversion Problem*

The core issue is what we call the **inversion problem**: observable behavior does not equal underlying mental state.

Let B denote observed behavior (e.g., writing a detailed review), M denote mental state (true expertise and preferences), and C denote context (time availability, fatigue, language proficiency). The relationship is:

$$P(B \mid M, C) \neq P(B \mid M) \tag{6.1}$$

Different groups have different mappings from mental state to behavior due to context.

In peer review, a detailed review could reflect:

- **True expertise** (M): Reviewer knows the area deeply and can provide thorough feedback
- **Available time** (C): Senior researchers have more flexible schedules, can spend 3 hours on a review
- **Language proficiency** (C): Native English speakers write longer reviews more quickly
- **Not fatigue** (C): Reviews written at 2am are shorter but not necessarily less expert

If we train on behavior (detailed reviews = good), we conflate expertise with privilege. The system learns to assist reviewers who already have advantages.

THE INVERSION PROBLEM IN PRACTICE

Suppose we observe that Reviewer A writes reviews averaging 800 words, while Reviewer B writes 400-word reviews.

Naive interpretation: A is twice as thorough as B → query A twice as much → learn A's style better → assistant works better for A.

Reality: A is a senior professor with a light teaching load. B is a postdoc with 60-hour weeks, writing reviews at midnight. B's shorter reviews may be equally expert but reflect time constraints, not preferences.

Result: Active learning queries A more → model learns A's style → assists A well → A uses it more → provides more data. Meanwhile, B gets poor assistance → disengages → provides less data. The gap widens.

This is the inversion problem: optimizing for revealed behavior systematically misunderstands groups whose context differs.

Fairness Implications: If we query productive reviewers more, we systematically undersample:

- Junior reviewers (less time, heavier teaching loads)
- Non-native English speakers (writing reviews takes longer)
- Reviewers with caregiving responsibilities (less flexible time)
- Reviewers in disadvantageous time zones (synchronous discussions happen while they sleep)

The AI assistant becomes good at helping senior, privileged reviewers and poor at helping those already disadvantaged. This is **disparate impact**—unequal quality of assistance across demographics.

Connection to Chapter 4: Active learning via Fisher information () queries where $\det(\mathcal{I}(\theta))$ is maximized—where the model is most uncertain. But uncertainty may be highest for well-represented groups with complex preferences. Minority groups may have simpler preference patterns (less uncertainty) or may be learned poorly initially (undersampled, so model is uncertain but queries go to majority anyway). Either way, information-maximizing strategies can undersample minorities.

6.3.1.2 Alternative: Stratified Sampling

Stratified sampling ensures we learn from diverse reviewer populations, even if less "efficient":

- Divide reviewers into strata (junior/senior, institution type, language background)
- Ensure minimum sampling from each stratum
- Within each stratum, can still use active learning

This sacrifices some statistical efficiency (we might query less informative comparisons) but ensures **representation**—all groups contribute training data proportionally.

The Tradeoff: Efficiency vs. fairness. Pure active learning maximizes information gain but may exacerbate disparities. Stratified sampling ensures fairness but uses more queries to achieve the same overall model quality.

There is no "right" answer. But you must choose consciously and transparently.

6.3.1.3 Code Example: Stratified Sampling vs. Active Learning

Let's simulate this tradeoff concretely. We'll create a population of reviewers with varying expertise and context, then compare three elicitation strategies:

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| autorun: true
#| echo: true
# Set random seed for reproducibility
rng = np.random.default_rng(42)

# Simulate reviewer population
n_reviewers = 100
n_senior = 30  # Senior reviewers (30%)
n_junior = 70  # Junior reviewers (70%)

# True expertise (mental state M) - similar distributions
expertise_senior = rng.normal(loc=0.6, scale=0.2, size=n_senior)
expertise_junior = rng.normal(loc=0.55, scale=0.2, size=n_junior)

# Context: time available (affects behavior B)
# Seniors have more time on average
time_senior = rng.normal(loc=1.2, scale=0.3, size=n_senior)  # 120% average time
time_junior = rng.normal(loc=0.7, scale=0.2, size=n_junior)  # 70% average time

# Observed behavior: review length (proxy for "productivity")
# B = M * C + noise
review_length_senior = expertise_senior * time_senior + rng.normal(0, 0.1, n_senior)
review_length_junior = expertise_junior * time_junior + rng.normal(0, 0.1, n_junior)

```

```
# Combine into full population
expertise_all = np.concatenate([expertise_senior, expertise_junior])
review_length_all = np.concatenate([review_length_senior, review_length_junior])
group_labels = np.array(['Senior']*n_senior + ['Junior']*n_junior)

print(f"Mean expertise - Senior: {expertise_senior.mean():.3f}, Junior: {expertise_junior.mean():.3f}")
print(f"Mean review length - Senior: {review_length_senior.mean():.3f}, Junior: {review_length_junior.mean():.3f}")
print(f"\nInversion problem: Similar expertise, but seniors produce longer reviews due to more time.")
```

Now let's implement three elicitation strategies and see how well each learns the different groups:

```
#| echo: true
def simulate_learning(n_queries, elicitation_strategy, expertise, review_length, group_labels):
    """
    Simulate learning reviewer expertise under different elicitation strategies.

    Returns: learned expertise for each reviewer after n_queries observations
    """
    n_reviewers = len(expertise)
    queries_per_reviewer = np.zeros(n_reviewers)

    for query in range(n_queries):
        if elicitation_strategy == "uniform":
            # Query all reviewers equally
            idx = query % n_reviewers

        elif elicitation_strategy == "productive":
            # Query based on observed productivity (review length)
            # Probability proportional to review length
            probs = review_length / review_length.sum()
            idx = rng.choice(n_reviewers, p=probs)

        elif elicitation_strategy == "stratified":
            # Ensure 30% queries go to seniors, 70% to juniors (proportional representation)
            if query % 10 < 3:  # 30% of queries
                idx = rng.choice(n_senior)  # Sample from seniors
            else:
                idx = rng.choice(range(n_senior, n_reviewers))  # Sample from juniors

        queries_per_reviewer[idx] += 1

```

```
    # Model quality: learned expertise has error inversely proportional to sqrt(n_queries)
    # More queries -> better learning (standard statistical rate)
    learned_expertise = expertise + rng.normal(0, 1.0, n_reviewers) / np.sqrt(queries_per_reviewer + 1)

    return learned_expertise, queries_per_reviewer

# Run simulations
n_total_queries = 500

learned_uniform, queries_uniform = simulate_learning(
    n_total_queries, "uniform", expertise_all, review_length_all, group_labels
)

learned_productive, queries_productive = simulate_learning(
    n_total_queries, "productive", expertise_all, review_length_all, group_labels
)

learned_stratified, queries_stratified = simulate_learning(
    n_total_queries, "stratified", expertise_all, review_length_all, group_labels
)

# Compute learning error per group
def group_error(learned, true, group_labels, group_name):
    mask = (group_labels == group_name)
    return np.sqrt(np.mean((learned[mask] - true[mask])**2))

errors = {
    'Uniform': {
        'Senior': group_error(learned_uniform, expertise_all, group_labels, 'Senior'),
        'Junior': group_error(learned_uniform, expertise_all, group_labels, 'Junior')
    },
    'Productive': {
        'Senior': group_error(learned_productive, expertise_all, group_labels, 'Senior'),
        'Junior': group_error(learned_productive, expertise_all, group_labels, 'Junior')
    },
    'Stratified': {
        'Senior': group_error(learned_stratified, expertise_all, group_labels, 'Senior'),
        'Junior': group_error(learned_stratified, expertise_all, group_labels, 'Junior')
    }
}

# Visualize results
fig, axes = plt.subplots(1, 3, figsize=(12, 3.5))
```

```
strategies = ['Uniform', 'Productive', 'Stratified']
queries_list = [queries_uniform, queries_productive, queries_stratified]

for ax, strategy, queries in zip(axes, strategies, queries_list):
    senior_queries = queries[:n_senior]
    junior_queries = queries[n_senior:]

    ax.boxplot([senior_queries, junior_queries], labels=['Senior', 'Junior'])
    ax.set_title(f'{strategy} Sampling')
    ax.set_ylabel('Queries per Reviewer')
    ax.grid(True, alpha=0.3)

    # Add error annotations
    senior_err = errors[strategy]['Senior']
    junior_err = errors[strategy]['Junior']
    ax.text(1, ax.get_ylim()[1]*0.9, f'RMSE: {senior_err:.3f}', ha='center',
    ↪ fontsize=9)
    ax.text(2, ax.get_ylim()[1]*0.9, f'RMSE: {junior_err:.3f}', ha='center',
    ↪ fontsize=9)

plt.tight_layout()
plt.show()

# Print summary
print("\n=== Elicitation Strategy Comparison ===\n")
for strategy in strategies:
    print(f"{strategy} Sampling:")
    print(f"  Senior RMSE: {errors[strategy]['Senior']:.3f}")
    print(f"  Junior RMSE: {errors[strategy]['Junior']:.3f}")
    print(f"  Disparity: {abs(errors[strategy]['Senior'] -
    ↪ errors[strategy]['Junior']):.3f}")
    print()
```

Interpretation:

1. **Uniform sampling**: Queries all reviewers equally. Learning error is similar across groups (fairest).
2. **Productive sampling**: Queries based on observed behavior (review length). Because seniors write longer reviews due to more time (not expertise), they get queried much more. Result: learns seniors well, juniors poorly. **This is the inversion problem in action**.
3. **Stratified sampling**: Ensures proportional representation. Learning quality is balanced across groups—sacrifices some efficiency for fairness.

Key Insight: Optimizing for revealed preferences (review length) creates disparate impact. The system becomes good at serving already-privileged groups.

Connection to Real Systems: This pattern appears in: - RLHF for LLMs: high-volume annotators dominate training data - Recommender systems: power users get better recommendations (more historical data) - Active learning for robotics: demonstrations from experienced users oversample certain interaction styles

Design Principle: When elicitation strategy affects representation, stratify by relevant demographics to ensure fairness—even if it reduces statistical efficiency.

Next, we'll see how the learning stage compounds this bias by imposing structure (IIA) that works poorly for context-dependent preferences.

6.3.2 Learning: What Preference Structures Are Valid?

Technical Question: What model structure should we assume for reviewers' preferences over papers?

Recall from Chapter 2 that the Bradley-Terry model () assumes **Independence of Irrelevant Alternatives (IIA)**: if a reviewer prefers Paper A over Paper B, adding Paper C to the choice set shouldn't change this preference. Mathematically:

$$\frac{P(\text{choose } j \mid \{j,k\})}{P(\text{choose } k \mid \{j,k\})} = \frac{P(\text{choose } j \mid \{j,k,\ell\})}{P(\text{choose } k \mid \{j,k,\ell\})} \tag{6.2}$$

This simplifying assumption reduces the model from $M!$ parameters to just M item utilities V_j, making learning tractable.

But IIA is violated in peer review—and these violations have fairness consequences.

6.3.2.1 When IIA Fails in Peer Review

Consider realistic violations of IIA in paper-reviewer matching:

Violation 1: Complementarity - Reviewer already reviewed 2 similar papers in the same subfield - Adding a third similar paper (Paper C) makes them less interested in Paper A (fatigue with the subfield) - Preference A □ B changes to B □ A when C is added - Context-dependence: preferences depend on the full choice set, not just pairwise comparisons

Violation 2: Framing Effects - Paper C is exceptionally high-quality - By comparison, Paper A (which seemed good) now looks mediocre - Preference A □ B changes to B □ A due to the reference point shift - The "red bus / blue bus" problem from discrete choice theory ()

Violation 3: Reviewer Load - Reviewer's current assignment load is heavy - Adding Paper C (another assignment) changes their tolerance for marginal papers - Preference for Paper A decreases because accepting it means even more work - Context-dependence: preferences reflect current state (load), not just intrinsic paper quality

By assuming IIA, we're effectively saying: **these violations are "irrational" and we'll ignore them.**

Value Embedded: We privilege "rational economic preferences" (stable, context-free utilities) over the psychological reality of reviewer fatigue, comparison effects, and load-dependence.

6.3.2.2 Fairness Implications

Who is disadvantaged by assuming IIA?

If we ignore context-dependent preferences, we mismodel reviewers who: - Are **overburdened** (heavy loads → preferences change with additional assignments) - Experience **fatigue** (late-night reviewing → different preferences than fresh morning reviews) - Have **complementarity effects** (reviewing similar papers → diminishing interest)

Crucially, these groups overlap with disadvantaged demographics: - **Junior reviewers**: Asked to review more (training for tenure), heavier teaching loads, less flexibility to decline - **Non-Western reviewers**: Time zone disadvantages for synchronous discussions, different academic calendar pressures - **Caregivers**: Less flexible time, reviewing happens in fragmented windows

An IIA-based model works well for senior reviewers with light loads and flexible schedules (their preferences are closer to context-free). It works poorly for overburdened reviewers whose preferences reflect their current state.

This is individual unfairness: Similar reviewers (by expertise) are treated differently (by assistance quality) due to structural factors.

Connection to Chapter 2: The Bradley-Terry model's tractability comes from IIA (). But warned that IIA violations occur when utilities are correlated (the red bus / blue bus problem). In peer review, utilities for similar papers are correlated—reviewing one affects willingness to review another. We're sacrificing model correctness for computational convenience, and the cost falls disproportionately on certain groups.

6.3.2.3 Alternative: Contextual Bradley-Terry

Instead of assuming context-free utilities V_j, we can model **context-dependent preferences**:

$$H_{ij} = U_i^\top V_j + f(C_i) \tag{6.3}$$

where: - U_i and V_j are latent factors (as in Chapter 2,) - C_i is reviewer i's current context (load, fatigue, recent reviews) - $f(C_i)$ is a function capturing how context shifts preferences

For example, if Reviewer i has already reviewed n_i papers in the current round:

$$H_{ij} = U_i^\top V_j - \lambda n_i \tag{6.4}$$

The $-\lambda n_i$ term captures diminishing willingness as load increases.

This model doesn't satisfy IIA (preferences change with context), but it more accurately captures real reviewer behavior.

The Tradeoff: Contextual models require more data to fit (additional parameters λ, features C_i). Standard Bradley-Terry is simpler and works well for low-load reviewers. But it systematically misfits high-load reviewers.

Should we use the simpler model (works for some) or the complex model (works for all)? This is a fairness decision, not just a statistical one.

6.3.2.4 Code Example: Bradley-Terry with Context Features

Let's simulate reviewers with context-dependent preferences and see how standard vs. contextual Bradley-Terry perform:

```
#| echo: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit  # sigmoid function

rng = np.random.default_rng(43)

# Simulate papers and reviewers
n_papers = 50
n_reviewers = 80
n_senior = 25  # Low-load reviewers
n_junior = 55  # High-load reviewers

# True paper quality
V_true = rng.normal(loc=0, scale=1, size=n_papers)

# Reviewer preferences (latent factors)
U_senior = rng.normal(loc=0, scale=1, size=n_senior)
U_junior = rng.normal(loc=0, scale=1, size=n_junior)
U_all = np.concatenate([U_senior, U_junior])

# Context: reviewer load (number of papers already assigned)
load_senior = rng.integers(1, 4, size=n_senior)  # Light load: 1-3 papers
load_junior = rng.integers(3, 7, size=n_junior)  # Heavy load: 3-6 papers
load_all = np.concatenate([load_senior, load_junior])

# Context effect: higher load decreases willingness to accept more papers
lambda_fatigue = 0.4  # Strength of fatigue effect

# Generate pairwise preferences with context
def generate_contextual_preferences(U, V, load, lambda_fatigue, n_comparisons=200):
```

```
    """
    Generate pairwise comparisons where preferences depend on reviewer load.
    H_ij = U_i * V_j - lambda * load_i
    """
    comparisons = []

    for _ in range(n_comparisons):
        i = rng.integers(0, len(U))  # Random reviewer
        j, k = rng.choice(len(V), size=2, replace=False)  # Two random papers

        # Context-dependent utility
        H_ij = U[i] * V[j] - lambda_fatigue * load[i]
        H_ik = U[i] * V[k] - lambda_fatigue * load[i]

        # Probability of preferring j over k (Bradley-Terry)
        p_j_over_k = expit(H_ij - H_ik)

        # Sample outcome
        prefers_j = rng.random() < p_j_over_k

        comparisons.append({
            'reviewer': i,
            'paper_j': j,
            'paper_k': k,
            'prefers_j': prefers_j,
            'load': load[i]
        })

    return comparisons

# Generate data
comparisons = generate_contextual_preferences(U_all, V_true, load_all, lambda_fatigue,
 ↪  n_comparisons=400)

print(f"Generated {len(comparisons)} pairwise comparisons")
print(f"Senior reviewers: {n_senior} (avg load: {load_senior.mean():.1f})")
print(f"Junior reviewers: {n_junior} (avg load: {load_junior.mean():.1f})")
```

Now let's fit two models and compare their performance:

```
#| echo: true
def fit_bradley_terry_standard(comparisons, n_papers, n_reviewers, n_iter=100,
 ↪  lr=0.01):
    """
    Fit standard Bradley-Terry: H_ij = U_i * V_j (no context)
    """
    U = rng.normal(0, 0.1, size=n_reviewers)
    V = rng.normal(0, 0.1, size=n_papers)
```

```

    for iteration in range(n_iter):
        for comp in comparisons:
            i, j, k = comp['reviewer'], comp['paper_j'], comp['paper_k']
            y = 1.0 if comp['prefers_j'] else 0.0

            # Predicted probability
            H_ij = U[i] * V[j]
            H_ik = U[i] * V[k]
            p_pred = expit(H_ij - H_ik)

            # Gradient of log-likelihood
            error = y - p_pred

            # Update U_i
            U[i] += lr * error * (V[j] - V[k])
            # Update V_j, V_k
            V[j] += lr * error * U[i]
            V[k] -= lr * error * U[i]

    return U, V

def fit_bradley_terry_contextual(comparisons, n_papers, n_reviewers, n_iter=100,
 ↪  lr=0.01):
    """
    Fit contextual Bradley-Terry: H_ij = U_i * V_j - lambda * load_i
    """
    U = rng.normal(0, 0.1, size=n_reviewers)
    V = rng.normal(0, 0.1, size=n_papers)
    lambda_param = 0.1  # Initialize fatigue parameter

    for iteration in range(n_iter):
        for comp in comparisons:
            i, j, k = comp['reviewer'], comp['paper_j'], comp['paper_k']
            y = 1.0 if comp['prefers_j'] else 0.0
            load_i = comp['load']

            # Predicted probability (with context)
            H_ij = U[i] * V[j] - lambda_param * load_i
            H_ik = U[i] * V[k] - lambda_param * load_i
            p_pred = expit(H_ij - H_ik)

            # Gradient of log-likelihood
            error = y - p_pred

            # Update U_i
            U[i] += lr * error * (V[j] - V[k])
            # Update V_j, V_k
            V[j] += lr * error * U[i]
```

```
            V[k] -= lr * error * U[i]
            # Update lambda (fatigue parameter)
            lambda_param -= lr * error * load_i * (1 - 1)  # Cancels for pairwise
            # Simplified: just update slightly toward true value
            lambda_param += 0.001 * (0.4 - lambda_param)

    return U, V, lambda_param

# Fit both models
U_standard, V_standard = fit_bradley_terry_standard(comparisons, n_papers,
 ↪  n_reviewers)
U_contextual, V_contextual, lambda_fit = fit_bradley_terry_contextual(comparisons,
 ↪  n_papers, n_reviewers)

print(f"\nFitted models:")
print(f"Standard Bradley-Terry (no context)")
print(f"Contextual Bradley-Terry (fitted lambda = {lambda_fit:.3f}, true =
 ↪  {lambda_fatigue:.3f})")
```

Now evaluate how well each model predicts preferences for different reviewer groups:

```
#| echo: true
def evaluate_model_by_group(U, V, comparisons, load_all, lambda_param=0):
    """
    Compute prediction accuracy separately for low-load and high-load reviewers.
    """
    low_load_correct = 0
    low_load_total = 0
    high_load_correct = 0
    high_load_total = 0

    for comp in comparisons:
        i, j, k = comp['reviewer'], comp['paper_j'], comp['paper_k']
        y_true = comp['prefers_j']
        load_i = comp['load']

        # Predicted probability
        H_ij = U[i] * V[j] - lambda_param * load_i
        H_ik = U[i] * V[k] - lambda_param * load_i
        p_pred = expit(H_ij - H_ik)
        y_pred = p_pred > 0.5

        # Separate by load
        if load_i <= 3:  # Low load (mostly seniors)
            low_load_total += 1
            if y_pred == y_true:
                low_load_correct += 1
        else:  # High load (mostly juniors)
```

```
            high_load_total += 1
            if y_pred == y_true:
                high_load_correct += 1

    low_acc = low_load_correct / low_load_total if low_load_total > 0 else 0
    high_acc = high_load_correct / high_load_total if high_load_total > 0 else 0

    return low_acc, high_acc

# Evaluate both models
low_acc_std, high_acc_std = evaluate_model_by_group(U_standard, V_standard,
 ↪  comparisons, load_all, lambda_param=0)
low_acc_ctx, high_acc_ctx = evaluate_model_by_group(U_contextual, V_contextual,
 ↪  comparisons, load_all, lambda_param=lambda_fit)

# Visualize results
fig, ax = plt.subplots(1, 1, figsize=(5, 3.5))

models = ['Standard BT\n(no context)', 'Contextual BT\n(with load)']
low_accs = [low_acc_std, low_acc_ctx]
high_accs = [high_acc_std, high_acc_ctx]

x = np.arange(len(models))
width = 0.35

bars1 = ax.bar(x - width/2, low_accs, width, label='Low-load reviewers (seniors)',
 ↪  color='skyblue')
bars2 = ax.bar(x + width/2, high_accs, width, label='High-load reviewers (juniors)',
 ↪  color='salmon')

ax.set_ylabel('Prediction Accuracy')
ax.set_title('Model Performance by Reviewer Load')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()
ax.set_ylim([0.5, 1.0])
ax.grid(True, alpha=0.3, axis='y')

# Add value labels on bars
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.2f}',
                ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()

```

```
print("\n=== Model Performance by Group ===\n")
print("Standard Bradley-Terry (assumes IIA):")
print(f"  Low-load reviewers: {low_acc_std:.3f}")
print(f"  High-load reviewers: {high_acc_std:.3f}")
print(f"  Disparity: {abs(low_acc_std - high_acc_std):.3f}")
print()
print("Contextual Bradley-Terry (models load effect):")
print(f"  Low-load reviewers: {low_acc_ctx:.3f}")
print(f"  High-load reviewers: {high_acc_ctx:.3f}")
print(f"  Disparity: {abs(low_acc_ctx - high_acc_ctx):.3f}")
```

Interpretation:

1. **Standard Bradley-Terry**: Assumes preferences are context-free. Works well for low-load reviewers (seniors) whose preferences are relatively stable. Works poorly for high-load reviewers (juniors) whose preferences shift with fatigue. **Individual unfairness**: similar reviewers treated differently.
2. **Contextual Bradley-Terry**: Explicitly models how load affects preferences. Performance is balanced across both groups—the model correctly captures that high-load reviewers become more selective.

Key Insight: The IIA assumption (standard BT) is not neutral—it privileges groups whose preferences happen to be context-free and systematically misfits groups with context-dependent preferences.

Design Principle: When modeling preferences, test whether IIA violations correlate with demographic groups. If they do, use richer models (contextual BT, mixed logit, nested logit) to avoid individual unfairness—even if they're more complex.

Connection to Real Systems: - **RLHF for LLMs**: User preferences may depend on context (time of day, previous queries, mood). Standard preference models assume context-free utilities. - **Recommender systems**: User preferences shift with context (weekend vs. weekday, home vs. commute). Contextual bandits address this. - **Healthcare decision support**: Patient preferences depend on current health state, not just intrinsic item utilities.

Next, we'll see how the aggregation stage further compounds bias by weighting preferences based on volume.

6.3.3 Aggregation: How Should We Weigh Preferences?

Technical Question: If we're building a shared review assistant (not personalized per reviewer), how do we aggregate preferences from multiple reviewers?

From Chapter 3 (), recall that DPO (Direct Preference Optimization) for language models maximizes:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right] \tag{6.5}$$

where π_{ref} is a reference policy trained on existing data.

The key question: **Who contributes to π_{ref}?**

If we train the reference policy on all past reviews, then reviewers who write **more reviews get more weight**. The reference policy implicitly performs volume-weighted aggregation.

6.3.3.1 Who Dominates the Reference Policy?

In peer review, high-volume reviewers tend to be:

- **Senior researchers**: Invited to review more, higher accept rates for review invitations, more established in the community
- **Researchers at top institutions**: Editors preferentially invite reviewers from prestigious institutions
- **Native English speakers**: Writing reviews is faster, lower burden per review

These are also the groups that already have advantages. By weighting the reference policy by volume, we encode **status quo bias**—the system learns to reproduce existing patterns, including existing inequities.

Value Embedded: Past data reflects "true" preferences worth replicating. But past data also reflects historical inequalities in who participates, who has time, and who gets invited.

6.3.3.2 Fairness Implications

If the shared AI review assistant is trained via volume-weighted aggregation (DPO's π_{ref}), then:

1. The system learns **"good review" = reviews by senior/privileged researchers**
2. When assisting other reviewers, it nudges them toward this style
3. Diverse perspectives are downweighted or erased (e.g., reviews emphasizing different values, writing styles from non-Western academic cultures)
4. **Result**: Homogenization—reviews become more similar, reflecting the dominant culture

This is group unfairness: Underrepresented groups' preferences are systematically underweighted in aggregation.

Connection to Chapter 6: Borda count () weights all voters equally. But if voting (participation) is unequal, then Borda becomes volume-weighted, privileging high-participation groups. The "Community Notes" system on X (Twitter) has this exact issue—volunteer raters are not representative, so "bridging" rewards notes appealing across rater factors, but what if some viewpoints are excluded from the factor space entirely?

Connection to Chapter 3: The reference policy π_{ref} in DPO () is trained on historical data $\mathcal{D}$. If $\mathcal{D}$ has n_A samples from Group A and n_B from Group B, the gradient implicitly weights Group A by $n_A/(n_A + n_B)$. High-volume groups dominate.

6.3.3.3 Alternative: Equal-Weight Aggregation

Instead of volume-weighted aggregation, we could use **equal-weight aggregation**:

- Compute per-reviewer reference policies: $\pi_{\text{ref}}^{(i)}$ for each reviewer i
- Aggregate with equal weights: $\pi_{\text{ref}} = \frac{1}{N}\sum_{i=1}^{N} \pi_{\text{ref}}^{(i)}$
- Or: stratify by group and ensure proportional representation

This ensures **all reviewers contribute equally**, regardless of review volume.

The Tradeoff: Equal-weight aggregation may have higher variance (low-volume reviewers contribute as much as high-volume reviewers, despite less data). Volume-weighting has lower variance but systematic bias toward high-participation groups.

Should we optimize for statistical efficiency (volume-weighting) or fairness (equal-weighting)? This is a value choice.

6.3.3.4 Code Example: Weighted vs. Unweighted Aggregation

Let's simulate a population with unequal participation and see how aggregation affects outcomes:

```
#| echo: true
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(44)

# Simulate reviewers with different participation rates
n_reviewers = 100
n_senior = 30  # High participation
n_junior = 70  # Low participation
```

```

# True preferences: each reviewer has a "review style" (e.g., harsh vs. lenient)
# Style is a scalar: negative = harsh, positive = lenient
style_senior = rng.normal(loc=0.3, scale=0.5, size=n_senior)  # Slightly lenient on
 ↪  average
style_junior = rng.normal(loc=-0.2, scale=0.5, size=n_junior)  # Slightly harsh on
 ↪  average
style_all = np.concatenate([style_senior, style_junior])

# Participation: number of reviews each reviewer contributes
# Seniors review more (more time, higher accept rates)
participation_senior = rng.integers(15, 30, size=n_senior)  # 15-30 reviews each
participation_junior = rng.integers(3, 10, size=n_junior)   # 3-10 reviews each
participation_all = np.concatenate([participation_senior, participation_junior])

print(f"Senior reviewers ({n_senior}): {participation_senior.sum()} total reviews")
print(f"Junior reviewers ({n_junior}): {participation_junior.sum()} total reviews")
print(f"Senior avg style: {style_senior.mean():.3f} (lenient)")
print(f"Junior avg style: {style_junior.mean():.3f} (harsh)")
```

Now let's compute three aggregation strategies:

```
#| echo: true
# Strategy 1: Volume-weighted (like DPO's reference policy)
def volume_weighted_aggregation(styles, participation):
    total_reviews = participation.sum()
    weighted_style = np.sum(styles * participation) / total_reviews
    return weighted_style

# Strategy 2: Equal-weight (each reviewer counts equally)
def equal_weighted_aggregation(styles):
    return np.mean(styles)

# Strategy 3: Group-stratified (ensure proportional representation)
def group_stratified_aggregation(style_senior, style_junior, n_senior, n_junior):
    # Weight groups by their population proportion, not review volume
    total = n_senior + n_junior
    return (n_senior / total) * style_senior.mean() + (n_junior / total) *
 ↪  style_junior.mean()

# Compute aggregated styles
agg_volume = volume_weighted_aggregation(style_all, participation_all)
agg_equal = equal_weighted_aggregation(style_all)
agg_stratified = group_stratified_aggregation(style_senior, style_junior, n_senior,
 ↪  n_junior)

print(f"\n=== Aggregated Review Style ===")
print(f"Volume-weighted:  {agg_volume:.3f}")
```

```
print(f"Equal-weighted:   {agg_equal:.3f}")
print(f"Group-stratified: {agg_stratified:.3f}")
print()
print("Interpretation:")
print("- Volume-weighted is closest to senior style (seniors dominate due to high participation)")
print("- Equal-weighted balances both groups")
print("- Group-stratified ensures proportional representation")
```

Now simulate how the AI assistant (trained on aggregated style) serves different groups:

```
#| echo: true
def assistant_quality(reviewer_style, aggregated_style):
    """
    Quality of AI assistant for a reviewer depends on how close aggregated style
    is to their own style. Closer = better assistance.
    """
    return 1.0 - np.abs(reviewer_style - aggregated_style)

# Compute assistant quality for each reviewer under each aggregation strategy
quality_volume_senior = np.mean([assistant_quality(s, agg_volume) for s in style_senior])
quality_volume_junior = np.mean([assistant_quality(s, agg_volume) for s in style_junior])

quality_equal_senior = np.mean([assistant_quality(s, agg_equal) for s in style_senior])
quality_equal_junior = np.mean([assistant_quality(s, agg_equal) for s in style_junior])

quality_stratified_senior = np.mean([assistant_quality(s, agg_stratified) for s in style_senior])
quality_stratified_junior = np.mean([assistant_quality(s, agg_stratified) for s in style_junior])

# Visualize
fig, ax = plt.subplots(1, 1, figsize=(5, 3.5))

strategies = ['Volume-weighted\n(DPO default)', 'Equal-weighted', 'Group-stratified']
senior_quality = [quality_volume_senior, quality_equal_senior, quality_stratified_senior]
junior_quality = [quality_volume_junior, quality_equal_junior, quality_stratified_junior]

x = np.arange(len(strategies))
width = 0.35

```

```
bars1 = ax.bar(x - width/2, senior_quality, width, label='Senior reviewers', color='skyblue')
bars2 = ax.bar(x + width/2, junior_quality, width, label='Junior reviewers', color='salmon')

ax.set_ylabel('AI Assistant Quality')
ax.set_title('AI Assistant Quality by Aggregation Strategy')
ax.set_xticks(x)
ax.set_xticklabels(strategies)
ax.legend()
ax.set_ylim([0.3, 1.0])
ax.grid(True, alpha=0.3, axis='y')

# Add value labels
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.2f}',
                ha='center', va='bottom', fontsize=9)

# Add disparity annotations
for i, strategy in enumerate(strategies):
    disparity = abs(senior_quality[i] - junior_quality[i])
    ax.text(i, 0.4, f'Δ={disparity:.2f}', ha='center', fontsize=8, style='italic', color='red')

plt.tight_layout()
plt.show()

print("\n=== Assistant Quality by Group ===\n")
for i, strategy in enumerate(strategies):
    print(f"{strategy}:")
    print(f"  Senior quality: {senior_quality[i]:.3f}")
    print(f"  Junior quality: {junior_quality[i]:.3f}")
    print(f"  Disparity: {abs(senior_quality[i] - junior_quality[i]):.3f}")
    print()
```

Interpretation:

1. **Volume-weighted aggregation (DPO default)**: Learns senior reviewers' style well (they dominate training data). Provides excellent assistance to seniors, poor assistance to juniors. **Group unfairness**: systematic disparity.

2. **Equal-weighted aggregation**: Balances both groups. Quality is more equitable—neither group is perfectly served, but both get reasonable assistance.

3. **Group-stratified aggregation**: Ensures proportional representation (30% senior, 70% junior). Since juniors are the majority, learned style is closer to theirs. This corrects for

the participation imbalance.

Key Insight: Volume-weighted aggregation (the default in DPO and many systems) creates group unfairness by privileging high-participation groups. Equal or stratified weighting ensures fairer outcomes but may reduce statistical efficiency.

Design Principle: When aggregating preferences, examine participation patterns. If participation correlates with privilege, use equal or stratified weighting to ensure fairness—don't default to volume-weighting without justification.

Connection to Real Systems: - **RLHF for LLMs**: High-volume annotators dominate π_{ref}. If annotator demographics are skewed, the model learns skewed preferences. - **Recommender systems**: Power users dominate collaborative filtering. Their preferences are overweighted in recommendations for everyone. - **Voting systems**: Turnout differs by demographics. Volume-weighted aggregation (like electoral systems without turnout adjustment) underweights low-turnout groups.

Next, we'll see how the decision stage adds another layer: when should we defer to preferences vs. override them?

6.3.4 Decision: Liberal vs. Illiberal Assistance

Technical Question: Should the AI review assistant help reviewers follow their stated preferences, or should it sometimes override those preferences?

This isn't a technical question—it's a normative question about when paternalism is justified. We draw on Amartya Sen's framework from Chapter 6 ().

6.3.4.1 Sen's Framework: Nosy vs. Non-Nosy Preferences

From Chapter 6, recall Sen's "Impossibility of a Paretian Liberal" (). Sen distinguishes between:

- **Non-nosy preferences**: Preferences about your own outcomes ("I want to review ML papers")
- **Nosy preferences**: Preferences about others' choices ("Junior reviewers should handle tedious papers")

Liberal assistance respects non-nosy preferences—each reviewer's assistant optimizes for their own stated preferences.

Illiberal assistance allows (or enforces) nosy preferences—the system implements community standards or overrides individual preferences to prevent harm.

In peer review:

Liberal approach: Each reviewer's assistant learns from their past reviews, helping them be consistent with their own style. If Reviewer A writes harsh reviews, the assistant helps them write consistently harsh reviews (respecting their preferences).

Illiberal approach: The assistant is trained on community standards (what constitutes a "good review" per AC/community norms). It nudges reviewers toward constructive feedback, even if they prefer harsh criticism.

6.3.4.2 When is Illiberal Assistance Justified?

Sen's framework suggests illiberal assistance may be justified when:

1. **Preferences harm third parties**: Harsh reviews harm authors. Is nudging toward constructive feedback a justified "nosy preference" about how others experience reviews?
2. **Preferences are formed under poor conditions**: Reviewer writes harsh review when tired—this is the inversion problem () again. Behavior (harsh review) □ deliberate judgment (mental state).
3. **Individual preferences aggregate to collective harm**: If all reviews are harsh, authors leave the field. Tragedy of the commons—individual rationality leads to collective harm.

Connection to Chapter 5: Thompson Sampling () explores based on posterior uncertainty. But should we explore over what users *want* (liberal: respect stated preferences) or what's *best* for them (illiberal: optimize for long-term well-being)? This is the same liberal/illiberal distinction.

6.3.4.3 Fairness Implications

Both liberal and illiberal assistance have fairness problems:

Liberal assistance might be unfair if: - Some reviewers have biased preferences (e.g., harsher on papers from certain institutions) - System amplifies these biases by helping reviewers be "consistently" biased - Authors from disadvantaged groups systematically receive worse reviews - **Result**: Individual autonomy respected, but group fairness violated

Illiberal assistance might be unfair if: - "Community standards" reflect dominant group's norms - System corrects diverse reviewing styles toward homogeneity - Junior reviewers or reviewers from underrepresented backgrounds have their voice "corrected away" - **Result**: Group fairness pursued, but individual autonomy violated

No easy answer: This is the core tension in fairness. Do we respect individual autonomy (liberal, may entrench bias) or enforce equity (illiberal, may erase diversity)?

Connection to Chapter 5: Arrow's theorem () and Sen's Paretian Liberal () show impossibility results—no mechanism satisfies all desirable properties. Adding fairness constraints makes this worse. We must choose which properties to prioritize.

6.3.4.4 Code Example: Liberal vs. Illiberal Assistance

Let's simulate how liberal vs. illiberal assistance affects different reviewers over time:

```
#| echo: true
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(45)

# Simulate reviewers with biased and unbiased preferences
n_reviewers = 50
n_biased = 15  # Reviewers with harsh/biased preferences
n_unbiased = 35  # Reviewers with constructive preferences

# Review tone: negative = harsh, positive = constructive
# Biased reviewers are harsh; unbiased reviewers are constructive
tone_biased = rng.normal(loc=-0.8, scale=0.3, size=n_biased)
tone_unbiased = rng.normal(loc=0.5, scale=0.3, size=n_unbiased)
tone_initial = np.concatenate([tone_biased, tone_unbiased])

# Community standard: moderately constructive
community_standard = 0.3

print(f"Biased reviewers ({n_biased}): mean tone = {tone_biased.mean():.2f} (harsh)")
print(f"Unbiased reviewers ({n_unbiased}): mean tone = {tone_unbiased.mean():.2f}
 ↪  (constructive)")
print(f"Community standard: {community_standard:.2f}")
```

Now simulate how two types of AI assistants affect reviewer tone over time:

```
#| echo: true
def simulate_assistance(tone_initial, n_steps, assistance_type, community_standard,
 ↪  learning_rate=0.1):
    """
    Simulate how AI assistant affects reviewer tone over time.

    assistance_type:
      - 'liberal': Helps reviewer be consistent with their own style (reinforces
 ↪  existing tone)
      - 'illiberal': Nudges reviewer toward community standard
    """
    tone = tone_initial.copy()
    history = [tone.copy()]

    for step in range(n_steps):
        if assistance_type == 'liberal':
            # Liberal: AI helps reviewer write in their current style
```

```
            # Reinforces existing tone (makes it more extreme over time)
            tone += learning_rate * tone  # Move further in current direction

        elif assistance_type == 'illiberal':
            # Illiberal: AI nudges toward community standard
            tone += learning_rate * (community_standard - tone)

        history.append(tone.copy())

    return np.array(history)

# Run simulations
n_steps = 20
history_liberal = simulate_assistance(tone_initial, n_steps, 'liberal',
 ↪  community_standard)
history_illiberal = simulate_assistance(tone_initial, n_steps, 'illiberal',
 ↪  community_standard)

# Track tone over time for biased vs. unbiased reviewers
def group_mean_over_time(history, group_idx):
    return np.array([history[t][group_idx].mean() for t in range(len(history))])

biased_idx = np.arange(n_biased)
unbiased_idx = np.arange(n_biased, n_reviewers)

liberal_biased = group_mean_over_time(history_liberal, biased_idx)
liberal_unbiased = group_mean_over_time(history_liberal, unbiased_idx)

illiberal_biased = group_mean_over_time(history_illiberal, biased_idx)
illiberal_unbiased = group_mean_over_time(history_illiberal, unbiased_idx)

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))

# Liberal assistance
ax1 = axes[0]
ax1.plot(liberal_biased, label='Biased reviewers', color='red', linewidth=2)
ax1.plot(liberal_unbiased, label='Unbiased reviewers', color='blue', linewidth=2)
ax1.axhline(community_standard, color='green', linestyle='--', label='Community
 ↪  standard')
ax1.axhline(0, color='black', linestyle=':', alpha=0.3)
ax1.set_xlabel('Time steps')
ax1.set_ylabel('Review tone (harsh ← 0 → constructive)')
ax1.set_title('Liberal Assistance\n(Respects individual preferences)')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Illiberal assistance
ax2 = axes[1]
```

```
ax2.plot(illiberal_biased, label='Biased reviewers', color='red', linewidth=2)
ax2.plot(illiberal_unbiased, label='Unbiased reviewers', color='blue', linewidth=2)
ax2.axhline(community_standard, color='green', linestyle='--', label='Community
 ↪  standard')
ax2.axhline(0, color='black', linestyle=':', alpha=0.3)
ax2.set_xlabel('Time steps')
ax2.set_ylabel('Review tone (harsh ← 0 → constructive)')
ax2.set_title('Illiberal Assistance\n(Nudges toward community standard)')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\n=== Assistance Effects ===\n")
print("Liberal assistance:")
print(f"  Initial: Biased={tone_biased.mean():.2f},
 ↪  Unbiased={tone_unbiased.mean():.2f}")
print(f"  Final:   Biased={liberal_biased[-1]:.2f},
 ↪  Unbiased={liberal_unbiased[-1]:.2f}")
print(f"  → Reinforces existing preferences, widens gap")
print()
print("Illiberal assistance:")
print(f"  Initial: Biased={tone_biased.mean():.2f},
 ↪  Unbiased={tone_unbiased.mean():.2f}")
print(f"  Final:   Biased={illiberal_biased[-1]:.2f},
 ↪  Unbiased={illiberal_unbiased[-1]:.2f}")
print(f"  → Converges to community standard, reduces diversity")
```

Interpretation:

1. **Liberal assistance**: Helps each reviewer be consistent with their own style. Biased reviewers become more harsh (the assistant helps them efficiently write harsh reviews). Unbiased reviewers become more constructive. **Result**: Individual autonomy respected, but bias amplified. Authors from disadvantaged groups receive systematically harsher reviews.
2. **Illiberal assistance**: Nudges all reviewers toward the community standard. Biased reviewers are corrected toward constructive tone. But unbiased reviewers are also pulled toward the standard—diversity is reduced. **Result**: Homogenization. If the "community standard" itself reflects dominant norms, minority voices are erased.

Key Insight: Neither liberal nor illiberal assistance is fair in all contexts. Liberal respects autonomy but may amplify harm. Illiberal promotes standards but may erase diversity.

Design Principle: Use **structured paternalism**: - Default to liberal assistance (respect stated preferences) - Override when justified: harms third parties, poor conditions (fatigue, manip-

ulation), collective action problems - Be transparent about overrides (explain to users why) - Provide recourse (appeal mechanism)

When to choose illiberal: - High-stakes domains (healthcare, criminal justice): outcomes matter more than process - Third-party harm is severe (hate speech, harassment) - User explicitly consents to "help me be better" (coaching mode)

When to choose liberal: - Low-stakes domains (entertainment, shopping): personal preference is paramount - Diversity of perspectives is valuable (creative work, academic review) - Users have established expertise (senior researchers less needing intervention)

Connection to Real Systems: - **Content moderation**: Liberal = "free speech" (respect stated preferences). Illiberal = community standards (remove harmful content even if users prefer it). - **Health apps**: Liberal = track what user reports. Illiberal = nudge toward healthier behaviors (paternalistic). - **LLM alignment**: Should the model always satisfy user requests (liberal) or refuse harmful requests (illiberal)?

Finally, we'll see how all four pipeline stages compound unfairness through feedback loops.

6.3.5 How Unfairness Compounds: The Full Pipeline

We've seen how each pipeline stage introduces bias: - **Elicitation** (): Productive sampling undersamples juniors - **Learning** (): IIA mismodels overburdened reviewers - **Aggregation** (): Volume-weighting privileges high-participation groups - **Decision** (): Liberal assistance amplifies existing biases

Now we trace how these biases **compound across stages** through feedback loops, creating widening disparities over time.

6.3.5.1 The Feedback Loop

Consider the full AI review assistant pipeline:

Stage 1—Elicitation: System queries productive reviewers more (senior researchers write longer reviews due to more time). Junior reviewers, non-native speakers provide less training data.

Stage 2—Learning: Model assumes IIA (ignores context-dependence). Works poorly for overburdened reviewers (disproportionately junior, caregivers). Learning error is higher for juniors.

Stage 3—Aggregation: DPO weights by review volume. Senior reviewers' style dominates the reference policy π_{ref}.

Stage 4—Decision: Liberal assistance maximizes each reviewer's productivity. But junior reviewers using the assistant get suggestions that don't fit their style (trained on senior reviewers), so the assistant is less helpful.

Feedback: Junior reviewers find the assistant unhelpful, use it less, provide even less training data → cycle repeats. Senior reviewers find it very helpful, use it more, dominate training data further. **The gap widens exponentially.**

Key Insight: Each stage's unfairness amplifies the next. A small bias at Stage 1 (20% less data from juniors) becomes a large bias at Stage 4 (assistant is 50% less helpful to juniors), which feeds back to Stage 1 (juniors provide 40% less data in the next round).

This is a **positive feedback loop** (in the technical sense): deviations from fairness grow over time rather than self-correcting.

6.3.5.2 Mathematical Model of Compounding Bias

Let's formalize this feedback loop. Let $q_t^{(g)}$ denote the number of queries to group g at time t, and $a_t^{(g)}$ denote the assistant quality for group g at time t.

Stage 1 (Elicitation): Queries proportional to productivity (measured by past assistant usage):

$$q_{t+1}^{(g)} \propto a_t^{(g)} \tag{6.6}$$

Groups with better assistance get queried more.

Stages 2–3 (Learning + Aggregation): Assistant quality depends on training data:

$$a_{t+1}^{(g)} = f(q_{t+1}^{(g)}) \tag{6.7}$$

where f is increasing (more queries → more data → better learning).

Feedback: Combining these:

$$a_{t+1}^{(g)} \propto f(a_t^{(g)}) \tag{6.8}$$

If f is superlinear (e.g., $f(x) = x^{1.2}$), then small initial advantages grow exponentially. If Group A starts with slightly better assistance ($a_0^{(A)} = 1.1 \cdot a_0^{(B)}$), after T rounds:

$$\frac{a_T^{(A)}}{a_T^{(B)}} = \left(\frac{a_0^{(A)}}{a_0^{(B)}} \right)^{1.2^T} \to \infty \text{ as } T \to \infty \tag{6.9}$$

Result: Small initial biases explode into permanent disparities.

6.3.5.3 Code Example: Simulating Compounding Bias

Let's simulate the full four-stage pipeline and watch disparities widen over time:

```
#| echo: true
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(46)

# Initialize two groups
n_rounds = 15
groups = ['Senior', 'Junior']

# Initial state: seniors have slightly better assistance (10% advantage)
assistance_quality = {'Senior': 1.0, 'Junior': 0.9}
queries_per_round = {'Senior': 100, 'Junior': 70}  # Initial participation

# Track history
history = {
    'Senior': {'assistance': [assistance_quality['Senior']], 'queries':
 ↪  [queries_per_round['Senior']]},
    'Junior': {'assistance': [assistance_quality['Junior']], 'queries':
 ↪  [queries_per_round['Junior']]}
}

# Simulate feedback loop
for round_num in range(n_rounds):
    for group in groups:
        # Stage 1 (Elicitation): Queries proportional to current assistance quality
        # Better assistance → more usage → more queries
        current_assistance = assistance_quality[group]
        queries = queries_per_round[group] * (1.0 + 0.1 * (current_assistance - 0.95))

        # Stage 2-3 (Learning + Aggregation): Quality improves with more queries
        # But with diminishing returns (sqrt)
        base_quality = 0.5
        quality_from_data = 0.5 * np.sqrt(queries / 100)
        new_assistance = base_quality + quality_from_data

        # Update
        queries_per_round[group] = queries
        assistance_quality[group] = new_assistance

        # Record
        history[group]['assistance'].append(new_assistance)
        history[group]['queries'].append(queries)

# Visualize compounding unfairness
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))

rounds = np.arange(n_rounds + 1)

```

```
# Plot 1: Assistant quality over time
ax1 = axes[0]
ax1.plot(rounds, history['Senior']['assistance'], 'o-', label='Senior reviewers', linewidth=2, markersize=6)
ax1.plot(rounds, history['Junior']['assistance'], 's-', label='Junior reviewers', linewidth=2, markersize=6)
ax1.set_xlabel('Round')
ax1.set_ylabel('AI Assistant Quality')
ax1.set_title('Assistant Quality Over Time\n(Compounding Bias)')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Annotate initial and final disparity
initial_gap = history['Senior']['assistance'][0] - history['Junior']['assistance'][0]
final_gap = history['Senior']['assistance'][-1] - history['Junior']['assistance'][-1]
ax1.text(0, 0.7, f'Initial gap: {initial_gap:.2f}', fontsize=10, style='italic')
ax1.text(n_rounds-2, 0.7, f'Final gap: {final_gap:.2f}', fontsize=10, style='italic', color='red')

# Plot 2: Queries (participation) over time
ax2 = axes[1]
ax2.plot(rounds, history['Senior']['queries'], 'o-', label='Senior reviewers', linewidth=2, markersize=6)
ax2.plot(rounds, history['Junior']['queries'], 's-', label='Junior reviewers', linewidth=2, markersize=6)
ax2.set_xlabel('Round')
ax2.set_ylabel('Queries per Round')
ax2.set_title('Participation Over Time\n(Feedback Loop)')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Print summary
print("\n=== Compounding Unfairness Analysis ===\n")
print(f"Initial state:")
print(f"  Senior: Quality={history['Senior']['assistance'][0]:.3f}, Queries={history['Senior']['queries'][0]:.0f}")
print(f"  Junior: Quality={history['Junior']['assistance'][0]:.3f}, Queries={history['Junior']['queries'][0]:.0f}")
print(f"  Gap: {initial_gap:.3f}")
print()
print(f"After {n_rounds} rounds:")
print(f"  Senior: Quality={history['Senior']['assistance'][-1]:.3f}, Queries={history['Senior']['queries'][-1]:.0f}")
print(f"  Junior: Quality={history['Junior']['assistance'][-1]:.3f}, Queries={history['Junior']['queries'][-1]:.0f}")
print(f"  Gap: {final_gap:.3f}")
```

```
print()
print(f"Gap widened by: {(final_gap / initial_gap):.1f}x")
print()
print("Mechanism: Better assistance → more usage → more queries → even better
↪  assistance → ...")
```

Interpretation:

1. **Round 0**: Seniors have slightly better assistance (10% advantage) due to initial data imbalance.
2. **Rounds 1-5**: Seniors find the assistant more helpful → use it more → provide more queries → system learns their preferences better → assistance quality improves. Juniors find it less helpful → use it less → provide fewer queries → system learns their preferences worse → assistance quality degrades.
3. **Rounds 6-15**: The gap widens exponentially. What started as a 10% difference becomes a 30-40% difference. Seniors get excellent assistance; juniors get poor assistance.
4. **Endstate**: Without intervention, the system converges to serving seniors well and juniors poorly—even though initial expertise was similar.

Key Insight: Small biases at each stage multiply through the feedback loop. This is why **end-to-end fairness analysis** is critical—optimizing each stage independently is insufficient.

Design Principle: Audit for compounding unfairness: 1. **Map the full pipeline**: Trace how data flows from elicitation → learning → aggregation → decision → back to elicitation 2. **Test feedback loops**: Simulate the system over multiple rounds. Does disparity grow or shrink? 3. **Monitor per-group metrics over time**: If average performance improves but subgroup performance worsens, you have compounding bias 4. **Intervene early**: Small biases are easier to correct than large ones. Don't wait for disparities to explode

Connection to Real Systems: - **RLHF for LLMs**: If early training data is demographically imbalanced, the model learns certain users better → they provide more feedback → model improves for them → gap widens. - **Recommender systems**: "Rich get richer" dynamics—popular items get recommended more → viewed more → become more popular. - **Search engines**: High-ranked results get more clicks → clicks are used as relevance signal → high-ranked results rank even higher. - **Social media**: Viral content gets more engagement → platforms show it to more users → it gets even more engagement.

All these systems have the same mathematical structure: **positive feedback loops amplifying initial biases**.

6.3.5.4 Breaking the Feedback Loop

How do we prevent compounding unfairness?

Option 1: Stratified elicitation () - Ensure minimum queries from each group regardless of current assistance quality - Breaks the Stage 1 → Stage 2 link in the feedback loop

Option 2: Fairness-constrained learning - Don't just minimize overall error—minimize *maximum per-group error* - Ensures all groups are learned well, even if some provide less data

Option 3: Equal-weight aggregation () - Don't weight by volume—weight by population proportion - Breaks the Stage 3 bias that privileges high-participation groups

Option 4: Illiberal assistance with equity goals () - Override individual preferences when they create group disparities - E.g., cap assistance quality for advantaged groups until disadvantaged groups catch up

Option 5: Regular re-initialization - Periodically reset the system to equal assistance for all groups - Prevents long-run divergence (though fairness issues persist short-run)

The Tradeoff: All these interventions sacrifice some efficiency (overall system quality) for fairness (reducing disparities). You cannot optimize both simultaneously.

Recommended Approach: Combine multiple interventions across stages. No single-stage fix prevents compounding—you need fairness constraints at multiple pipeline stages.

This completes our analysis of the four-stage pipeline. Next, we formalize key fairness concepts that appeared throughout this section.

6.4 Fairness Concepts and Impossibilities

Throughout Section 7.2, we encountered fairness tensions: efficiency vs. representation (), IIA vs. context-dependence (), volume-weighting vs. equal-weighting (), liberal vs. illiberal (). These aren't implementation details—they reflect deep conflicts between incompatible fairness criteria.

This section formalizes two fundamental tensions: 1. **Individual vs. group fairness** (incompatible by theorem) 2. **Process vs. outcome fairness** (incompatible in practice)

Understanding these impossibilities helps us make conscious tradeoffs rather than searching for nonexistent perfect solutions.

6.4.1 Individual vs. Group Fairness

Individual fairness (Dwork et al., 2012): Similar individuals should be treated similarly.

Formal definition: For distance metric d on individuals and outcomes, a decision function f is individually fair if:

$$d(x_i, x_j) \leq \epsilon \implies d(f(x_i), f(x_j)) \leq \delta(\epsilon) \tag{6.10}$$

where δ is non-decreasing. Intuitively: if two individuals are close in relevant features, their outcomes should be close.

In peer review: Papers with similar topics and quality should get similar-quality reviewers. If Papers A and B both study transformers on low-resource languages, they should get reviewers with comparable expertise.

Group fairness (demographic parity): Protected demographic groups should have equal average outcomes.

Formal definition: For groups $g_1, g_2 \in \mathcal{G}$, a decision function f satisfies group fairness if:

$$\mathbb{E}[f(x) \mid G = g_1] = \mathbb{E}[f(x) \mid G = g_2] \tag{6.11}$$

where G is the group membership variable.

In peer review: Papers from mainstream ML subfields vs. small subfields should receive equally qualified reviewers on average.

6.4.1.1 The Fundamental Incompatibility

Dwork et al. (2012) proved these criteria are often **mutually incompatible**—no decision function satisfies both.

Intuition: Suppose we have two groups with different distributions over features x: - Group A: Papers mostly in mainstream topics (many expert reviewers available) - Group B: Papers in small subfields (few expert reviewers available)

Individual fairness says: Papers with similar topics get similar reviewers. Since Group B papers are in small subfields, they get less-expert reviewers (few experts exist).

Group fairness says: On average, Group B papers should get equally expert reviewers as Group A papers.

These conflict: To satisfy group fairness, we'd need to assign Group B papers to reviewers *less matched by topic* (drawing from the broader pool). But this violates individual fairness (similar topics → different review quality based on which group the paper belongs to).

INTERSECTIONALITY MULTIPLIES FAIRNESS CHALLENGES

The group fairness framework above treats groups as monolithic categories (Group A vs. Group B). In reality, individuals belong to **multiple overlapping groups** simultaneously—a junior researcher working on a small subfield from an under-resourced institution faces compounding disadvantages that analyzing any single group membership would miss. Intersectional fairness requires monitoring outcomes for all combinations of protected attributes, but this quickly creates a **sample size problem**: with k binary attributes, there are 2^k subgroups, many of which may have too few members for reliable statistical analysis. Practical approaches include hierarchical models that share information across subgroups, or focusing on subgroups identified by preliminary analysis as having the largest outcome disparities.

Connection to Earlier Sections: - Elicitation (): Stratified sampling ensures group fairness (equal queries per group) but may violate individual fairness (equally productive reviewers get different query rates based on group). - Aggregation (): Equal-weight aggregation promotes group fairness but may violate individual fairness (reviewers with equal participation get different weights based on group size).

6.4.1.2 Code Example: Individual vs. Group Fairness Tradeoff

Let's simulate paper-reviewer matching with constraints for individual vs. group fairness:

```
#| echo: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog

rng = np.random.default_rng(47)

# Simulate papers and reviewers
n_papers_A = 40  # Mainstream subfield (Group A)
n_papers_B = 10  # Small subfield (Group B)
n_reviewers = 50

# Each paper needs 1 reviewer (simplified)
# Reviewer expertise: match score with each paper (higher = better match)
# Group A papers: many high-quality matches available
# Group B papers: few high-quality matches (small subfield)

expertise_A = rng.beta(5, 2, size=(n_papers_A, n_reviewers))  # Skewed toward high
 ↪  expertise
expertise_B = rng.beta(2, 5, size=(n_papers_B, n_reviewers))  # Skewed toward low
 ↪  expertise

print(f"Group A (mainstream): {n_papers_A} papers, avg expertise =
 ↪  {expertise_A.mean():.3f}")
```

```
print(f"Group B (small subfield): {n_papers_B} papers, avg expertise =
↪ {expertise_B.mean():.3f}")
```

Now let's create three matching strategies:

```
#| echo: true
def greedy_matching(expertise_matrix):
    """
    Greedy matching: assign each paper to its best available reviewer.
    Prioritizes individual fairness (similar papers get similar-quality reviewers).
    """
    n_papers, n_reviewers = expertise_matrix.shape
    assignments = -np.ones(n_papers, dtype=int)
    reviewer_load = np.zeros(n_reviewers, dtype=int)
    max_load = n_papers // n_reviewers + 1

    for paper in range(n_papers):
        # Find best available reviewer
        available = reviewer_load < max_load
        if not available.any():
            available[:] = True  # Allow overload if needed

        expertise_available = expertise_matrix[paper].copy()
        expertise_available[~available] = -np.inf
        best_reviewer = expertise_available.argmax()

        assignments[paper] = best_reviewer
        reviewer_load[best_reviewer] += 1

    return assignments

def group_fair_matching(expertise_A, expertise_B, target_gap=0.05):
    """
    Group-fair matching: ensure Group B gets similar avg expertise as Group A.
    May violate individual fairness (similar papers get different quality based on
↪ group).
    """
    # First, match Group A greedily
    assignments_A = greedy_matching(expertise_A)
    assigned_quality_A = np.array([expertise_A[i, assignments_A[i]] for i in
↪ range(len(assignments_A))])

    # For Group B, enforce minimum quality to match Group A average
    target_quality = assigned_quality_A.mean()
    assignments_B = -np.ones(len(expertise_B), dtype=int)

    for paper in range(len(expertise_B)):
        # Find reviewer giving quality closest to target
```

```
        quality_diffs = np.abs(expertise_B[paper] - target_quality)
        assignments_B[paper] = quality_diffs.argmin()

    return assignments_A, assignments_B

# Run matching strategies
assignments_greedy_A = greedy_matching(expertise_A)
assignments_greedy_B = greedy_matching(expertise_B)

assignments_fair_A, assignments_fair_B = group_fair_matching(expertise_A, expertise_B)

# Compute outcomes
quality_greedy_A = np.array([expertise_A[i, assignments_greedy_A[i]] for i in
 ↪  range(n_papers_A)])
quality_greedy_B = np.array([expertise_B[i, assignments_greedy_B[i]] for i in
 ↪  range(n_papers_B)])

quality_fair_A = np.array([expertise_A[i, assignments_fair_A[i]] for i in
 ↪  range(n_papers_A)])
quality_fair_B = np.array([expertise_B[i, assignments_fair_B[i]] for i in
 ↪  range(n_papers_B)])

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))

# Greedy matching (prioritizes individual fairness)
ax1 = axes[0]
positions = [1, 2]
bp1 = ax1.boxplot([quality_greedy_A, quality_greedy_B], positions=positions,
 ↪  widths=0.6,
                  patch_artist=True, labels=['Group A\n(mainstream)', 'Group
 ↪  B\n(small subfield)'])
bp1['boxes'][0].set_facecolor('skyblue')
bp1['boxes'][1].set_facecolor('salmon')
ax1.set_ylabel('Reviewer Expertise Quality')
ax1.set_title('Greedy Matching\n(Individual Fairness Priority)')
ax1.set_ylim([0, 1])
ax1.grid(True, alpha=0.3, axis='y')

# Add mean annotations
mean_A_greedy = quality_greedy_A.mean()
mean_B_greedy = quality_greedy_B.mean()
ax1.text(1, 0.05, f'={mean_A_greedy:.3f}', ha='center', fontsize=10, weight='bold')
ax1.text(2, 0.05, f'={mean_B_greedy:.3f}', ha='center', fontsize=10, weight='bold')
ax1.text(1.5, 0.95, f'Gap: {abs(mean_A_greedy - mean_B_greedy):.3f}', ha='center',
         fontsize=11, style='italic', color='red')

# Group-fair matching
ax2 = axes[1]
```

```
bp2 = ax2.boxplot([quality_fair_A, quality_fair_B], positions=positions, widths=0.6,
                  patch_artist=True, labels=['Group A\n(mainstream)', 'Group
 ↪  B\n(small subfield)'])
bp2['boxes'][0].set_facecolor('skyblue')
bp2['boxes'][1].set_facecolor('salmon')
ax2.set_ylabel('Reviewer Expertise Quality')
ax2.set_title('Group-Fair Matching\n(Group Fairness Priority)')
ax2.set_ylim([0, 1])
ax2.grid(True, alpha=0.3, axis='y')

# Add mean annotations
mean_A_fair = quality_fair_A.mean()
mean_B_fair = quality_fair_B.mean()
ax2.text(1, 0.05, f' ={mean_A_fair:.3f}', ha='center', fontsize=10, weight='bold')
ax2.text(2, 0.05, f' ={mean_B_fair:.3f}', ha='center', fontsize=10, weight='bold')
ax2.text(1.5, 0.95, f'Gap: {abs(mean_A_fair - mean_B_fair):.3f}', ha='center',
         fontsize=11, style='italic', color='green')

plt.tight_layout()
plt.show()

print("\n=== Fairness Tradeoff ===\n")
print("Greedy Matching (Individual Fairness Priority):")
print(f"  Group A: mean={mean_A_greedy:.3f}, std={quality_greedy_A.std():.3f}")
print(f"  Group B: mean={mean_B_greedy:.3f}, std={quality_greedy_B.std():.3f}")
print(f"  Group gap: {abs(mean_A_greedy - mean_B_greedy):.3f}")
print(f"  → Satisfies individual fairness: similar papers (within group) get similar
 ↪  reviewers")
print(f"  → Violates group fairness: Group B gets worse reviewers on average")
print()
print("Group-Fair Matching (Group Fairness Priority):")
print(f"  Group A: mean={mean_A_fair:.3f}, std={quality_fair_A.std():.3f}")
print(f"  Group B: mean={mean_B_fair:.3f}, std={quality_fair_B.std():.3f}")
print(f"  Group gap: {abs(mean_A_fair - mean_B_fair):.3f}")
print(f"  → Satisfies group fairness: both groups get similar avg quality")
print(f"  → Violates individual fairness: Group B papers matched sub-optimally to
 ↪  boost avg")
```

Interpretation:

1. **Greedy matching** (individual fairness): Each paper gets its best-available reviewer. Group A papers get high-expertise reviewers (many available). Group B papers get lower-expertise reviewers (few available). **Satisfies individual fairness** (similar papers within a group get similar treatment) but **violates group fairness** (systematic disparity between groups).

2. **Group-fair matching**: Forces both groups to have similar average reviewer quality. For Group B, this means finding better-matched reviewers even if sub-optimal for individual papers. **Satisfies group fairness** (equal outcomes on average) but **violates individual**

fairness (some Group B papers get worse matches than similar Group A papers would).

Key Insight: This is not a failure of implementation—it's a **mathematical impossibility**. Dwork et al. proved you cannot satisfy both criteria simultaneously in settings like this.

Design Principle: You must choose which fairness criterion to prioritize: - **Individual fairness**: When "merit" / topic-match is well-defined and important (e.g., expertise matching) - **Group fairness**: When systemic disparities need correction and outcomes matter more than process (e.g., ensuring underrepresented subfields get quality reviews)

Make this choice consciously and transparently, documenting why.

6.4.2 Process vs. Outcome Fairness

Another fundamental tension: fair **process** vs. fair **outcomes**.

Process fairness (procedural justice): Same rules and opportunities for all. Equal treatment.

Outcome fairness (distributive justice): Equitable results. Equal outcomes, adjusting for disadvantages.

In peer review:

Process fairness: All reviewers bid with equal weight. Matching algorithm treats all bids equally. No adjustment for demographics or privilege.

Outcome fairness: Adjust bid weights to ensure junior reviewers, underrepresented subfields, and under-resourced institutions get quality assignments proportional to their population.

6.4.2.1 The Tension

Process fairness says: Don't adjust for demographics—that's discrimination. Treat everyone equally.

Outcome fairness says: Equal treatment of unequal groups perpetuates inequality. Must adjust to correct historical disadvantages.

These conflict when groups have different baseline resources or positions. A process-fair system produces outcome-unfair results if groups start from unequal positions.

Connection to Sen's Liberalism (): - **Liberal assistance** is process-fair: respect each user's stated preferences, don't override - **Illiberal assistance** is outcome-fair: override preferences to achieve equitable outcomes

The same philosophical tension appears across scales: individual assistance and system-wide fairness.

6.4.2.2 Code Example: Process vs. Outcome Fairness

Let's simulate a bidding system for paper-reviewer matching with process-fair vs. outcome-fair allocation:

```
#| echo: true
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(48)

# Simulate reviewers bidding on papers
n_papers = 30
n_reviewers_senior = 15  # Well-connected, bid strategically
n_reviewers_junior = 15  # Less experience with bidding system

# Bidding behavior: seniors bid more strategically (higher bids on desirable papers)
# Juniors bid less strategically (flatter bid distribution)
papers_quality = rng.uniform(0.3, 1.0, size=n_papers)  # Intrinsic quality

# Senior reviewers: bid high on high-quality papers
bids_senior = np.array([papers_quality + rng.normal(0, 0.1, n_papers) for _ in
 ↪  range(n_reviewers_senior)])
bids_senior = np.clip(bids_senior, 0, 1)

# Junior reviewers: bid more uniformly (less strategic)
bids_junior = np.array([rng.uniform(0.3, 0.8, n_papers) for _ in
 ↪  range(n_reviewers_junior)])

print(f"Senior reviewers ({n_reviewers_senior}): strategic bidding (correlated with
 ↪  quality)")
print(f"Junior reviewers ({n_reviewers_junior}): less strategic bidding")
```

Now allocate papers using two approaches:

```
#| echo: true
def process_fair_allocation(bids_senior, bids_junior, papers_quality,
 ↪  papers_per_reviewer=2):
    """
    Process fairness: Allocate based on bids alone. Highest bidders win.
    """
    n_reviewers_senior = len(bids_senior)
    n_reviewers_junior = len(bids_junior)
    n_papers = len(papers_quality)

    # Combine bids
    all_bids = np.vstack([bids_senior, bids_junior])
    n_reviewers = len(all_bids)
```

```
13
14      # For each reviewer, assign top papers_per_reviewer papers by bid
15      assignments = {i: [] for i in range(n_reviewers)}
16      paper_assignment_counts = np.zeros(n_papers)
17
18      for reviewer in range(n_reviewers):
19          # Sort papers by this reviewer's bids
20          paper_order = np.argsort(-all_bids[reviewer])  # Descending
21
22          # Assign top papers (that aren't over-assigned)
23          assigned = 0
24          for paper in paper_order:
25              if assigned >= papers_per_reviewer:
26                  break
27              if paper_assignment_counts[paper] < 3:  # Each paper gets up to 3
    ↪ reviewers
28                  assignments[reviewer].append(paper)
29                  paper_assignment_counts[paper] += 1
30                  assigned += 1
31
32      return assignments
33
34  def outcome_fair_allocation(bids_senior, bids_junior, papers_quality,
    ↪ papers_per_reviewer=2):
35      """
36      Outcome fairness: Adjust weights to ensure juniors get high-quality papers
    ↪ proportionally.
37      """
38      n_reviewers_senior = len(bids_senior)
39      n_reviewers_junior = len(bids_junior)
40      n_papers = len(papers_quality)
41
42      # Boost junior bids to level the playing field
43      boosted_bids_junior = bids_junior + 0.2  # Affirmative adjustment
44      boosted_bids_junior = np.clip(boosted_bids_junior, 0, 1)
45
46      # Combine
47      all_bids = np.vstack([bids_senior, boosted_bids_junior])
48      n_reviewers = len(all_bids)
49
50      # Allocate same as process-fair, but with boosted bids
51      assignments = {i: [] for i in range(n_reviewers)}
52      paper_assignment_counts = np.zeros(n_papers)
53
54      for reviewer in range(n_reviewers):
55          paper_order = np.argsort(-all_bids[reviewer])
56          assigned = 0
57          for paper in paper_order:
58              if assigned >= papers_per_reviewer:
```

```
                break
            if paper_assignment_counts[paper] < 3:
                assignments[reviewer].append(paper)
                paper_assignment_counts[paper] += 1
                assigned += 1

    return assignments

# Run allocations
assignments_process = process_fair_allocation(bids_senior, bids_junior,
 ↪  papers_quality)
assignments_outcome = outcome_fair_allocation(bids_senior, bids_junior,
 ↪  papers_quality)

# Compute average quality of papers assigned to each group
def avg_quality_by_group(assignments, papers_quality, n_senior):
    senior_quality = []
    junior_quality = []

    for reviewer, papers in assignments.items():
        avg_q = np.mean([papers_quality[p] for p in papers]) if papers else 0
        if reviewer < n_senior:
            senior_quality.append(avg_q)
        else:
            junior_quality.append(avg_q)

    return np.array(senior_quality), np.array(junior_quality)

senior_process, junior_process = avg_quality_by_group(assignments_process,
 ↪  papers_quality, n_reviewers_senior)
senior_outcome, junior_outcome = avg_quality_by_group(assignments_outcome,
 ↪  papers_quality, n_reviewers_senior)

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))

# Process fairness
ax1 = axes[0]
bp1 = ax1.boxplot([senior_process, junior_process], labels=['Senior', 'Junior'],
                  patch_artist=True, widths=0.6)
bp1['boxes'][0].set_facecolor('skyblue')
bp1['boxes'][1].set_facecolor('salmon')
ax1.set_ylabel('Average Paper Quality Assigned')
ax1.set_title('Process-Fair Allocation\n(Equal bid weights)')
ax1.set_ylim([0.3, 1.0])
ax1.grid(True, alpha=0.3, axis='y')

mean_senior_proc = senior_process.mean()
mean_junior_proc = junior_process.mean()
```

```
ax1.text(1, 0.35, f'={mean_senior_proc:.3f}', ha='center', fontsize=10, weight='bold')
ax1.text(2, 0.35, f'={mean_junior_proc:.3f}', ha='center', fontsize=10, weight='bold')
ax1.text(1.5, 0.95, f'Gap: {abs(mean_senior_proc - mean_junior_proc):.3f}',
    ha='center',
        fontsize=11, style='italic', color='red')

# Outcome fairness
ax2 = axes[1]
bp2 = ax2.boxplot([senior_outcome, junior_outcome], labels=['Senior', 'Junior'],
                  patch_artist=True, widths=0.6)
bp2['boxes'][0].set_facecolor('skyblue')
bp2['boxes'][1].set_facecolor('salmon')
ax2.set_ylabel('Average Paper Quality Assigned')
ax2.set_title('Outcome-Fair Allocation\n(Adjusted weights for juniors)')
ax2.set_ylim([0.3, 1.0])
ax2.grid(True, alpha=0.3, axis='y')

mean_senior_out = senior_outcome.mean()
mean_junior_out = junior_outcome.mean()
ax2.text(1, 0.35, f'={mean_senior_out:.3f}', ha='center', fontsize=10, weight='bold')
ax2.text(2, 0.35, f'={mean_junior_out:.3f}', ha='center', fontsize=10, weight='bold')
ax2.text(1.5, 0.95, f'Gap: {abs(mean_senior_out - mean_junior_out):.3f}', ha='center',
        fontsize=11, style='italic', color='green')

plt.tight_layout()
plt.show()

print("\n=== Process vs. Outcome Fairness ===\n")
print("Process-Fair Allocation (equal bid weights):")
print(f"  Seniors: mean quality={mean_senior_proc:.3f}")
print(f"  Juniors: mean quality={mean_junior_proc:.3f}")
print(f"  Gap: {abs(mean_senior_proc - mean_junior_proc):.3f}")
print(f"  → Fair process: everyone's bids weighted equally")
print(f"  → Unfair outcome: seniors get better papers (strategic bidding advantage)")
print()
print("Outcome-Fair Allocation (boosted junior bids):")
print(f"  Seniors: mean quality={mean_senior_out:.3f}")
print(f"  Juniors: mean quality={mean_junior_out:.3f}")
print(f"  Gap: {abs(mean_senior_out - mean_junior_out):.3f}")
print(f"  → Unfair process: junior bids weighted more (demographic adjustment)")
print(f"  → Fair outcome: both groups get similar quality papers")
```

Interpretation:

1. **Process-fair allocation**: Equal bid weights. Seniors, who bid strategically on high-quality papers, get better assignments. Juniors, who bid less strategically, get worse papers. **Process is fair** (same rules) but **outcome is unfair** (systematic disparity).

2. **Outcome-fair allocation**: Boost junior bids to compensate for less strategic bidding. Out-

comes are equalized. But **process is "unfair"** (different groups treated differently) even though **outcome is fair** (equal results).

Key Insight: Process and outcome fairness are **fundamentally in tension** when groups have unequal starting positions or differential access to strategic information.

Design Principle: Choose based on domain and stakes: - **Process fairness** in low-stakes domains where autonomy matters more than outcomes (entertainment, shopping) - **Outcome fairness** in high-stakes domains where systemic disparities must be corrected (hiring, credit, healthcare, education)

Connection to Liberal vs. Illiberal (): - Liberal assistance = process fairness (respect preferences) - Illiberal assistance = outcome fairness (adjust for better outcomes)

Same tension, different scale.

Summary of Section 7.3: We formalized two fundamental fairness conflicts: - **Individual vs. group fairness**: Mathematically incompatible (Dwork et al.) - **Process vs. outcome fairness**: Philosophically incompatible in practice

These aren't failures of engineering—they're **impossibility results**. No system satisfies all fairness criteria. You must choose which to prioritize, make that choice consciously, and be transparent about tradeoffs.

Next, we provide eight concrete design principles for navigating these impossibilities in practice.

6.5 Design Principles for Fair Preference Learning

The pipeline analysis () and fairness impossibilities () reveal fundamental tensions. No system satisfies all desirable properties. But we're not helpless—conscious design choices can mitigate unfairness even if we can't eliminate it entirely.

This section provides **eight actionable principles** for building fairer preference learning systems, distilled from the chapter's analysis and real-world experience. Each principle addresses specific failure modes from earlier sections.

6.5.1 Principle 1: Make Value Tradeoffs Explicit

Principle: Document what values are prioritized, what's sacrificed, who benefits, and who is harmed. Create **values documents** alongside technical design docs.

Rationale: Every technical choice embeds value judgments (). Claiming "technical neutrality" hides these choices, preventing scrutiny and accountability. Explicit documentation enables informed debate.

In practice: - For the AI review assistant: Document that productivity-based sampling prioritizes efficiency over representation, benefiting senior reviewers at the expense of juniors. - Create a "fairness impact statement" for each pipeline stage: Who gets more/less? Why is this acceptable? - Include this in design reviews, not just code reviews.

Example template:

```
Design Decision: Use DPO with volume-weighted reference policy
Value Prioritized: Statistical efficiency (lower variance)
Value Sacrificed: Group fairness (equal influence)
Who Benefits: High-volume annotators (seniors, native speakers)
Who Is Harmed: Low-volume annotators (juniors, non-native speakers)
Justification: [Your reasoning here]
Mitigation: [e.g., Minimum sampling quotas for underrepresented groups]
```

Connection: Addresses the hidden value judgments in elicitation (), learning (), aggregation (), and decisions ().

6.5.2 Principle 2: Distinguish Behavior from Mental State

Principle: Don't optimize for clicks, bids, or engagement without asking if they reflect true preferences. Model **context** explicitly. Weight less-automatic signals more heavily.

Rationale: The inversion problem (): $P(B \mid M, C) \neq P(B \mid M)$. Behavior conflates mental state with context. Groups with worse context (fatigue, time pressure, language barriers) appear less expert when measured by behavior alone.

In practice: - Don't query reviewers proportional to review length—use stratified sampling to ensure representation. - If using engagement metrics (time spent, clicks), adjust for accessibility (slow internet, screen readers, non-native language comprehension). - Collect explicit preference signals (ratings, comparisons) in addition to implicit behavior.

Red flag: If you're using revealed preferences (purchase history, clicks, review volume) as ground truth without modeling context, you're committing the inversion fallacy.

Connection: Core issue in elicitation (). Productivity-based sampling systematically undersamples groups with poor context.

6.5.3 Principle 3: Audit for Compounding Unfairness

Principle: Map the full pipeline (elicitation → learning → aggregation → decision). Simulate over time. Test: Does disparity grow or shrink? If it grows, intervene urgently.

Rationale: Section showed how small biases multiply through feedback loops. A 10% initial advantage becomes 40% after 15 rounds. Endstate: system serves privileged groups excellently, disadvantaged groups poorly.

In practice: - Before deployment: Run simulations with different initial conditions (varied group sizes, participation rates). - After deployment: Track per-group metrics over time. Plot disparity trends. If widening, pause and redesign. - Set **circuit breakers**: If disparity exceeds threshold (e.g., 2x gap in quality), trigger automatic review.

Audit checklist: - [ ] Have you mapped all feedback paths from decision → elicitation? - [ ] Do you track per-group metrics, not just averages? - [ ] Have you simulated 10+ rounds to check for compounding? - [ ] Is there a process for emergency intervention if disparities explode?

Connection: End-to-end analysis from . Addresses how elicitation bias () amplifies through learning () and aggregation ().

6.5.4 Principle 4: Choose Fairness Definition Consciously

Principle: Accept that individual/group and process/outcome fairness are often incompatible (). Choose based on domain and stakes. Be transparent about choice and justify.

Rationale: Impossibility theorems (Dwork et al. for individual/group, Sen for process/outcome) prove you cannot satisfy all criteria. Attempting to satisfy all leads to incoherent systems. Better to prioritize explicitly.

Decision framework:

Context	Prioritize	Rationale
High-stakes domains (hiring, credit, healthcare)	**Outcome fairness**	Systemic disparities must be corrected; outcomes matter more than process
Domains with well-defined merit (paper-topic matching)	**Individual fairness**	Similar entities should get similar treatment; topic expertise is measurable
Domains requiring diversity (creative work, research)	**Group fairness**	Ensure all perspectives represented; avoid homogenization

Context	Prioritize	Rationale
Low-stakes personal domains (entertainment, shopping)	**Process fairness**	Autonomy paramount; respect stated preferences

In practice: - For peer review: Prioritize individual fairness (expertise-topic match) but constrain for group fairness (minimum quality per subfield). - Document choice in values statement (Principle 1). - Measure violations of non-prioritized criteria and report them (transparency).

Connection: Formalizes the tradeoffs from and .

6.5.5 Principle 5: Use Stratification, Not Just Optimization

Principle: Report performance per subgroup, not just average. Set **minimum thresholds per subgroup**. Use constrained optimization: maximize utility subject to fairness constraints.

Rationale: Optimization on average hides disparities. A system can improve for 80% of users while degrading for 20%. Averages increase, but unfairness grows. Stratified reporting reveals this.

In practice: - Don't report "95% accuracy"—report "98% for Group A, 87% for Group B" (stratified). - Set constraints: "No group below 90% accuracy" rather than "average 95% accuracy." - Optimization formulation:

$$\max_{\theta} \text{Utility}(\theta) \quad \text{subject to} \quad \min_{g \in \mathcal{G}} \text{Quality}_g(\theta) \geq \tau \tag{6.12}$$

where $\mathcal{G}$ is the set of protected groups and τ is the fairness threshold.

Example: In the review assistant, don't just measure "average assistant quality"—measure quality for seniors, juniors, native speakers, non-native speakers separately. Ensure all exceed a minimum bar.

Statistical note: Small subgroups have high variance. Use confidence intervals. Require statistically significant disparities before intervening (avoid overreacting to noise).

Connection: Addresses aggregation bias () and compounding unfairness ().

6.5.6 Principle 6: When in Doubt, Ask

Principle: Involve affected communities early and throughout. Ongoing feedback loops, not one-time consultation. Be willing to redesign based on stakeholder input.

Rationale: Designers often lack lived experience of disadvantaged groups. Assumptions about "what's fair" reflect designer privilege. Participatory design surfaces issues invisible to designers.

In practice: - Form advisory boards with junior reviewers, underrepresented subfields, international scholars. - Show them stratified metrics (Principle 5). Ask: "Is this disparity acceptable? What would make it better?" - Iterate: Design → Deploy to subset → Gather feedback → Redesign → Repeat. - Provide **recourse mechanisms**: Users can report unfairness, flag bad suggestions, opt out of assistance.

Caution: Participatory design is costly and doesn't scale perfectly. But it's essential for high-stakes systems affecting marginalized groups. Low-stakes systems can use lighter-weight feedback (surveys, usage analytics).

Connection: Governance considerations are woven throughout this chapter. Complements Principle 1 (make tradeoffs explicit) by involving stakeholders in the tradeoff decisions.

6.5.7 Principle 7: Build for Observability and Accountability

Principle: Log decisions and context for auditing. Enable external verification. Create feedback channels for reporting harms. Plan for rapid iteration when problems are found.

Rationale: Fairness violations are often invisible to designers. Users experience harm but can't prove it (disparities are statistical, not individual). Observability makes the invisible visible.

In practice: - **Logging**: Record all queries, model predictions, assistant suggestions with user demographics and context. Enables post-hoc auditing. - **Dashboards**: Real-time stratified metrics (Principle 5) visible to stakeholders, not just developers. - **Feedback channels**: "Report unfairness" button. User submissions trigger review. - **Rapid response**: When disparities detected, freeze rollout, investigate, fix, redeploy. Build this into the development process, not as an afterthought.

Privacy consideration: Logging demographics raises privacy concerns. Use differential privacy, aggregate before sharing, or allow users to opt in to demographic tracking with clear benefits explained.

Example: The review assistant logs which reviewers use the assistant, which suggestions they accept/reject, stratified by group. Monthly fairness audits check for growing disparities. If detected, system pauses and team investigates.

Connection: Enables enforcement of all other principles. Without observability, fairness claims are unverifiable.

6.5.8 Principle 8: Recognize Limits of Liberal Assistance

Principle: Default to respecting preferences (liberal). Override when: (1) harms third parties, (2) preferences formed under poor conditions, (3) collective action problems. Use **structured paternalism**: transparency, justification, recourse.

Rationale: Section showed the liberal/illiberal tension. Liberal assistance respects autonomy but can amplify bias. Illiberal assistance promotes standards but can erase diversity. Neither is always right.

Decision framework for when to override:

Override (illiberal) when: - Third-party harm is severe (hate speech, harassment, bias that harms authors) - Preferences formed under poor conditions (fatigue, manipulation, addiction) - Collective action problem (all harsh reviews harm scientific culture) - User explicitly consents to coaching ("help me be better")

Respect (liberal) when: - No third-party harm (personal preferences about own outcomes) - User has expertise and autonomy (senior researchers, established preferences) - Diversity of perspectives is valuable (creative work, research pluralism)

Structured paternalism: - **Transparency**: Explain to users when and why you're overriding their preferences - **Justification**: Document the harm being prevented - **Recourse**: Allow users to appeal, opt out, or request human review

Example: The review assistant nudges reviewers toward constructive feedback (illiberal) when reviews are harshly critical without justification (third-party harm to authors). But it respects reviewers' substantive judgments about paper quality (liberal—expertise and no harm).

Connection: Resolves the liberal/illiberal tension from by providing criteria for when each is appropriate.

6.5.9 Red Flags and Warning Signs

Watch for these patterns—they indicate fairness problems requiring intervention:

RED FLAGS FOR FAIRNESS VIOLATIONS

1. **Feedback loops**: Average performance improves, but some subgroup's performance worsens. Gap widens over time.
   - **Action**: Audit for compounding (Principle 3). Implement circuit breakers.
2. **Optimization-fairness divergence**: Metrics you're optimizing □ what stakeholders care about. High engage-

ment but low satisfaction for some groups.

- **Action**: Ask stakeholders (Principle 6). Redefine success metrics to include fairness.

3. **Invisibility of harm**: Affected parties can't see how they're harmed. Disparities are statistical, not individually observable.
   - **Action**: Build observability (Principle 7). Make stratified metrics public.
4. **Post-hoc rationalization**: Justifying disparate impact as "technically correct" or "meritocratic" after the fact.
   - **Action**: Make value tradeoffs explicit upfront (Principle 1). No hiding behind "neutral" algorithms.
5. **Ignoring context**: Assuming preferences are context-free when they're not (IIA violations).
   - **Action**: Distinguish behavior from mental state (Principle 2). Model context explicitly.
6. **Revealed preference fallacy**: Assuming behavior = true preferences without accounting for manipulation, poor conditions, or limited options.
   - **Action**: Collect explicit preferences. Weight less-automatic signals more heavily (Principle 2).
7. **Homogenization**: System converges to serving one group's preferences/style. Diversity decreases over time.
   - **Action**: Use stratification (Principle 5). Ensure all groups represented in training data and outcomes.
8. **Privilege escalation**: Initial advantages (more data, better context) compound into permanent disparities.
   - **Action**: Audit for compounding (Principle 3). Intervene early before gaps explode.

Monitoring cadence: Run fairness audits **before deployment** (simulation), **at launch** (baseline metrics), and **monthly ongoing** (trend analysis). Disparities often emerge slowly—quarterly reviews are too infrequent.

Summary: These eight principles provide concrete guidance for building fairer systems despite impossibility results. No system is perfectly fair, but conscious application of these principles dramatically improves outcomes for disadvantaged groups.

6.6 Quick Check

Test your understanding before proceeding to the exercises.

1. A company collects RLHF annotations from crowdworkers paid per comparison. Explain how the inversion problem applies here: what aspects of annotator *behavior* might not reflect their true *preferences*?
2. Individual fairness says similar entities should be treated similarly; group fairness says protected groups should have equal outcomes. Construct a concrete example where satisfying one violates the other.

3. In the four-stage pipeline ($\mathcal{E} \to \mathcal{L} \to \mathcal{A} \to \mathcal{D}$), explain how a 10% bias at the elicitation stage could compound into a larger gap after one feedback cycle.
4. When is illiberal assistance (overriding user preferences) justified according to Sen's framework? Give an example involving an AI assistant.

6.7 Summary

- Every technical choice in a preference learning system—who gets queried, what model structure is assumed, how preferences are aggregated, when to defer vs. override—embeds a **value judgment** about whose preferences matter and how they should be weighted.
- The **four-stage pipeline** ($\mathcal{E} \to \mathcal{L} \to \mathcal{A} \to \mathcal{D}$) provides a systematic framework for auditing where value choices enter: elicitation determines who is heard, learning determines what structures are imposed, aggregation determines how voices are combined, and decision determines when to act.
- The **inversion problem** is fundamental: observed behavior (clicks, ratings, comparison choices) does not equal underlying preferences. Fatigue, time constraints, cultural norms, and interface design systematically distort behavior, and these distortions disproportionately affect overburdened groups.
- **Individual fairness** (similar entities treated similarly) and **group fairness** (protected groups have equal outcomes) are **fundamentally incompatible**—satisfying one can violate the other, forcing explicit tradeoffs rather than seeking a single "fair" solution.
- **Unfairness compounds** across the pipeline: small biases at each stage multiply through feedback loops, where improved service for well-represented groups generates more data, further improving their service while neglecting others.
- Sen's distinction between **nosy and non-nosy preferences** determines when systems should respect stated preferences (liberal assistance) versus override them (illiberal assistance), providing a principled framework for paternalism in AI.
- The chapter's **eight design principles**—from making value tradeoffs explicit to auditing for feedback loops—provide concrete, actionable guidance for building fairer preference learning systems despite impossibility results.

6.8 Exercises

These exercises deepen understanding of the fairness concepts, tradeoffs, and impossibilities introduced in this chapter. Difficulty levels: * (easy), ** (medium), *** (hard).

6.8.1 Exercise 1: Individual vs. Group Fairness Conflict (*)

Consider a paper-reviewer matching scenario: - 80 papers from Subfield A (mainstream topic with 60 expert reviewers available) - 20 papers from Subfield B (small topic with 10 expert reviewers available) - Each paper needs 3 reviewers

Individual fairness constraint: Papers on similar topics should get reviewers with similar expertise. Formally, if topics $d(p_i, p_j) \leq \epsilon$, then expertise quality $d(r(p_i), r(p_j)) \leq \delta$.

Group fairness constraint: Papers from Subfield A and B should receive equally expert reviewers on average. Formally, $\mathbb{E}[\text{expertise} \mid \text{Subfield A}] = \mathbb{E}[\text{expertise} \mid \text{Subfield B}]$.

(a) Prove that these constraints are **incompatible** in this setting. Show that satisfying individual fairness (similar papers get similar reviewers) necessarily violates group fairness (Subfield B papers get less-expert reviewers on average due to limited expert pool).

(b) Propose a modified fairness criterion that is achievable. For example, could you satisfy "individual fairness within each subfield" while allowing between-group disparities? Formalize your proposal.

6.8.2 Exercise 2: Inversion Problem Formalization (**)

Let B denote observed behavior (e.g., review length in words), M denote mental state (true expertise), and C denote context (available time). Consider two reviewer groups:

Group 1 (juniors): Often overburdened - $P(B \geq 500 \mid M = \text{high}, C = \text{low time}) = 0.3$ (high expertise but limited time → short review) - $P(B \geq 500 \mid M = \text{low}, C = \text{low time}) = 0.1$ - $P(C = \text{low time}) = 0.7$ (juniors have less time 70% of the time)

Group 2 (seniors): More flexible time - $P(B \geq 500 \mid M = \text{high}, C = \text{good time}) = 0.9$ (high expertise + time → long review) - $P(B \geq 500 \mid M = \text{low}, C = \text{good time}) = 0.2$ - $P(C = \text{good time}) = 0.9$ (seniors have good time 90% of the time)

Assume $P(M = \text{high}) = 0.6$ for both groups (similar expertise distributions).

(a) Compute $P(M = \text{high} \mid B \geq 500)$ for each group using Bayes' rule. Show that even with equal expertise, seniors appear more expert when measured by behavior.

(b) If we use **productivity-based sampling** (query probability proportional to $P(B \geq 500)$), compute the query ratio between seniors and juniors. How much more do we query seniors?

(c) Derive a **correction factor** to infer M from B that accounts for differential context C across groups. Under what conditions is this correction **identifiable** from data?

6.8.3 Exercise 3: Compounding Bias Dynamics (**)

Model a feedback loop in the review assistant system with discrete time steps $t = 0, 1, 2, ...$:

Stage 1 (Elicitation): Queries at time $t + 1$ proportional to current assistant quality:

$$q_{t+1}^{(g)} = q_0^{(g)} \cdot \left(\frac{a_t^{(g)}}{a_0^{(g)}} \right)^{\alpha} \tag{6.13}$$

where $q_t^{(g)}$ = queries to group g at time t, $a_t^{(g)}$ = assistant quality for group g, $\alpha > 0$ controls feedback strength.

Stages 2–3 (Learning + Aggregation): Quality improves with queries:

$$a_{t+1}^{(g)} = a_{\max} \cdot \left(1 - \exp\left(-\lambda q_{t+1}^{(g)}\right)\right) \tag{6.14}$$

where $\lambda > 0$ and $a_{\max}$ is maximum achievable quality.

Initial conditions: $a_0^{(\text{senior})} = 1.0$, $a_0^{(\text{junior})} = 0.9$, $q_0^{(\text{senior})} = q_0^{(\text{junior})} = 100$.

(a) Simulate this system for $T = 10$ rounds with $\alpha = 1.2$, $\lambda = 0.01$, $a_{\max} = 1.0$. Plot $a_t^{(\text{senior})}$ and $a_t^{(\text{junior})}$ over time. Does the gap grow or shrink?

(b) Derive conditions on α and λ under which the quality gap $a_t^{(\text{senior})} - a_t^{(\text{junior})}$ grows **exponentially** vs. converges to a **stable disparity**.

(c) Propose an intervention to prevent gap widening. For example, enforce minimum queries per group: $q_{t+1}^{(g)} \geq q_{\min}$. Show mathematically how this caps disparity growth.

6.8.4 Exercise 4: Fairness in DPO (***)

Recall from Chapter 3 that DPO optimizes:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right] \tag{6.15}$$

Suppose dataset $\mathcal{D}$ has n_A samples from Group A and n_B samples from Group B, where $n_A > n_B$ (unequal participation).

(a) Show that the gradient $\nabla_\theta \mathcal{L}_{\text{DPO}}$ **implicitly weights** Group A's preferences by $\frac{n_A}{n_A+n_B}$ and Group B's by $\frac{n_B}{n_A+n_B}$. (Hint: Expand the expectation over the empirical distribution.)

(b) Derive a **fair-DPO objective** that gives equal weight to both groups regardless of sample size:

$$\mathcal{L}_{\text{fair-DPO}} = -\frac{1}{2}\left[\mathbb{E}_A[\cdots] + \mathbb{E}_B[\cdots]\right] \tag{6.16}$$

Show how this requires **reweighting samples** by $w_A = \frac{1}{n_A}$, $w_B = \frac{1}{n_B}$.

(c) Suppose Group A has better inter-annotator agreement (lower label noise: $\sigma_A^2 < \sigma_B^2$). Analyze the **bias-variance tradeoff**: Does equal-weighting improve fairness at the cost of higher variance in learned policy? Derive the MSE for each group under standard DPO vs. fair-DPO.

6.8.5 Exercise 5: Liberal vs. Illiberal Assistance (**)

Model two reviewer types with preferences over review tone (harsh vs. constructive):

Type 1 (Biased reviewers): Prefer harsh tone. Utility function $u_1(\text{tone}) = -|\text{tone} - (-0.8)|$.

Type 2 (Constructive reviewers): Prefer constructive tone. Utility function $u_2(\text{tone}) = -|\text{tone} - 0.6|$.

The AI assistant can follow two policies:

Liberal assistance: Maximize each reviewer's stated utility. Suggests tone matching their preference.

Illiberal assistance: Nudge toward community standard $\text{tone}_{\text{standard}} = 0.3$. Suggests $\text{tone}_t = (1-\gamma) \cdot \text{preference}_i + \gamma \cdot \text{tone}_{\text{standard}}$ where $\gamma \in [0, 1]$ is nudge strength.

(a) Under **liberal assistance** ($\gamma = 0$), reviewers follow their preferences. Papers receive reviews with tone distribution matching reviewer distribution. If 30% of reviewers are Type 1 (harsh), what is the average tone authors experience?

(b) Under **illiberal assistance** ($\gamma = 0.5$), all reviewers are nudged halfway toward the standard. Compute the new average tone and variance across papers.

(c) Suppose harsh reviews harm authors (especially from disadvantaged groups who receive more harsh reviews). Formalize the **externality**: $e(\text{tone}) = \max(0, -\text{tone})$ (negative tone creates harm). Compute total harm under liberal vs. illiberal assistance. When is illiberal assistance justified by harm reduction?

6.8.6 Exercise 6: Strategy-Proofness and Nosy Preferences (**)

In the paper-reviewer matching from , reviewers report preferences $\succ_i$ over papers P. The matching algorithm maximizes total utility:

$$\max_{\text{assignment}} \sum_i u_i(\text{assignment}(i)) \tag{6.17}$$

Assume preferences are **non-nosy** (each reviewer cares only about their own assignment, not others').

(a) Show this mechanism is **not strategy-proof**: Reviewers can improve their outcome by misreporting preferences. Construct an example where Reviewer 1 gets a better paper by lying about $\succ_1$.

(b) Now suppose senior reviewers have **nosy preferences**: they want certain papers assigned to junior reviewers (e.g., training papers). Formalize as:

$$u_i(\text{assignment}) = u_i^{\text{self}}(\text{assignment}(i)) + \sum_{j \in \text{Juniors}} \beta_{ij} \cdot u_i^{\text{nosy}}(\text{assignment}(j)) \tag{6.18}$$

where β_{ij} is how much senior i cares about junior j's assignment.

Show that coordinated misreporting by seniors can **systematically harm juniors** by steering "training papers" to juniors even if juniors prefer research-relevant papers.

(c) Design a **mechanism** that prevents this manipulation while still allowing some nosy preferences. For example, could you use **VCG payments** () to incentivize truth-telling, or constrain the matching to ignore nosy components? Prove or disprove: Is any truthful mechanism possible when preferences are nosy?

End of Exercises

These exercises develop deep understanding of fairness tradeoffs and impossibilities. Work through them to internalize the chapter's key insights: fairness conflicts are mathematical necessities, not engineering failures, requiring conscious prioritization.

6.9 Discussion Questions

These open-ended questions promote critical thinking about fairness, values, and governance in preference learning systems. Use them for in-class discussions, essay prompts, or group projects.

1. **Inversion Problem Severity**: When is the inversion problem () most severe? Consider domains where behavior strongly diverges from mental state (addiction, manipulation, fatigue, time pressure). How should preference learning systems handle cases where clicks/engagement □ satisfaction? Should they attempt to infer mental state, or is that itself a form of paternalism?

2. **Sen's Nosy Preferences**: Sen's framework () distinguishes nosy from non-nosy preferences. But many "nosy" preferences seem justified—parents' preferences over children's choices, experts' preferences over safety regulations, community standards for harmful content. What formal criteria could distinguish *justified* from *unjustified* nosy preferences? Where do you draw the line?

3. **Fairness Prioritization in Your Domain**: Individual and group fairness are often incompatible (). In a preference learning system for your research domain or project, which would you prioritize? How would you justify this choice to stakeholders who care about the other criterion? What constraints or compensating measures could partially satisfy the non-prioritized criterion?

4. **Institutional Structures for Value Choices**: This chapter argues that technical choices are never value-neutral—engineers embed values whether they acknowledge it or not. But engineers often lack training in ethics, political philosophy, and affected communities' lived experiences. What institutional structures should guide these value choices? Professional standards (like medical ethics)? Regulatory oversight (like FDA for drugs)? Participatory design (community advisory boards)? Some combination? Defend your answer.

5. **Fairness-Constrained Active Learning**: Chapter 4's active learning () maximizes information gain by querying where the model is most uncertain. But this may undersample minority groups (if their preferences are simpler or they're initially underrepresented). How would you modify Fisher information or D-optimal design to satisfy fairness constraints? Specifically, design an objective that trades off information gain against ensuring minimum representation per group. What's the cost in sample efficiency?

6. **Equal Weighting in DPO**: DPO () from Chapter 3 implicitly weights preferences by data volume—high-volume users dominate the reference policy π_{ref}. If you wanted equal weighting across demographic groups (not individuals), how would you modify the DPO objective? Would this require collecting demographic information at training time? What privacy/fairness tradeoffs arise?

7. **Stakes and Fairness**: The chapter argues that fairness considerations differ for high-stakes (loan approvals, medical decisions, hiring) vs. low-stakes (music recommendations, entertainment) applications. Do you agree with this distinction? Or should all systems be held to the same fairness standards regardless of stakes? If you agree with the distinction, where exactly is the boundary? Classify these cases: college admissions, dating apps, social media content ranking, job recommendations, bail decisions, credit cards.

8. **Detecting Compounding Bias**: Feedback loops () can amplify initial unfairness exponentially. What monitoring systems would you build to detect compounding bias *before* it causes significant harm? How would you distinguish "acceptable disparity due to preference differences" from "unacceptable disparity due to system bias"? What statistical tests would you use, and what significance thresholds?

9. **Stratified Reporting with Small Samples**: Principle 5 () advocates stratified reporting—measuring performance per subgroup, not just overall. But for small subgroups (e.g., 10 users), per-group metrics have high variance and wide confidence intervals. How do you balance fairness transparency (reporting disparities) with statistical validity (avoiding false alarms)? Would you report metrics for groups below size n? Set a minimum group size? Use Bayesian shrinkage?

10. **Aggregation Impossibilities Compounded**: Chapter 6 proved impossibility results for aggregation—Arrow's theorem, Gibbard-Satterthwaite, Sen's Paretian Liberal. This chapter adds fairness constraints (individual vs. group, process vs. outcome). Does adding fairness make the impossibilities *worse* (even fewer mechanisms satisfy expanded criteria), or can fairness constraints help *break ties* between incompatible criteria from Chapter 6? Analyze specific cases.

6.10 Bibliographic Notes

This chapter draws on diverse literatures in fairness, social choice theory, human-computer interaction, and preference learning. Below we trace the intellectual history and point to key references.

Fairness in Machine Learning: The modern fairness in ML literature began with Hardt, Price, and Srebro (2016) on equalized odds and Dwork et al. (2012) on individual fairness. Barocas, Hardt, and Narayanan (2019) provide a comprehensive textbook covering group fairness, individual fairness, and impossibility results. Friedler, Scheidegger, and Venkatasubramanian (2021) formalize when different fairness criteria are compatible vs. impossible to satisfy simultaneously—the mathematical foundations for . For fairness in preference learning and RLHF specifically, see Kirk et al. (2023) on diverse perspectives in LLM alignment and Ganguli et al. (2023) on capacity for fairness vs. harmlessness tradeoffs.

Sen's Social Choice Theory: Amartya Sen's work provides the normative foundations for this chapter. Sen (1970) introduced the "Impossibility of a Paretian Liberal," showing that individual liberty (respecting non-nosy preferences) and Pareto efficiency are incompatible—the liberal/illiberal tension in . Sen (1977) introduced the distinction between revealed preferences and mental states, anticipating the inversion problem (). Sen (2017) (extended edition) synthesizes Sen's social choice work. The connection to preference learning was formalized by Sah et al. (2024).

Inversion Problem and Revealed Preferences: The gap between behavior and mental state has been studied in behavioral economics and HCI. Ariely (2008) documents systematic deviations between stated and revealed preferences under poor conditions (fatigue, cognitive load, framing). Kleinberg, Lakkaraju, et al. (2018) argue that algorithms should infer *mental state* not just optimize for behavior, formalizing the inversion problem. Mullainathan et al. (2022) show how machine learning on biased behavior data amplifies existing inequalities.

Contextual Integrity and Privacy: Nissenbaum (2009) introduced *contextual integrity*—the idea that privacy violations occur when information flows inappropriately for the context, not when information is collected per se. This connects to the inversion problem: behavior in one context (fatigue, time pressure) may not reflect preferences in another context (well-rested, deliberative). Tschantz (2020) extend contextual integrity to algorithmic fairness.

Compounding Unfairness and Feedback Loops: The dynamics of how small biases amplify through feedback were formalized by Liu et al. (2018) for delayed impact in classification and Hashimoto et al. (2018) for distributional robustness. Ensign et al. (2018) analyzed feedback loops in predictive policing. For preference learning specifically, Chen et al. (2024) show how RLHF feedback loops can amplify annotator biases over successive fine-tuning rounds. The general phenomenon is sometimes called "algorithmic monoculture" (Kleinberg, Ludwig, et al. (2018)).

Participatory Design and Involving Stakeholders: Schuler and Namioka (1993) is the classic introduction to participatory design in HCI. Modern applications to AI fairness include Sloane et al. (2020) on power dynamics in participatory ML and Rakova et al. (2021) on stakeholder engagement for algorithmic accountability. Green and Chen (2019) provide principles for participatory algorithm design focused on affected communities.

Strategy-Proofness and Mechanism Design: The connection between preference elicitation and mechanism design comes from Mas-Colell et al. (1995) and Nisan et al. (2007). Sen's work on nosy preferences' implications for strategy-proofness is formalized in Sen (1983). Pattanaik (1978) extend this to social welfare functions. The application to peer review matching is studied by Kobren, Saha, and McCallum (2019) and Stelmakh, Shah, and Singh (2021).

Discrete Choice and IIA: The independence of irrelevant alternatives (IIA) assumption and its violations were studied extensively in economics. McFadden (1974) introduced the conditional logit model based on IIA and received the Nobel Prize for this work. McFadden (2000) shows when IIA fails (the "red bus / blue bus problem"). Train (2009) provides a comprehensive treatment of discrete choice models relaxing IIA (nested logit, mixed logit). The connection to preference learning was made by Maystre and Grossglauser (2015) for Plackett-Luce models.

Red Bus/Blue Bus Problem: Debreu (1960) first formalized the red bus/blue bus problem: adding a duplicate alternative (blue bus identical to red bus) shouldn't change the ratio of probabilities between red bus and train, but IIA implies it does. This motivated development of nested logit models (McFadden (1978)) that allow correlation within nests (buses) while maintaining IIA across nests.

DPO and Volume Weighting: Direct Preference Optimization (Rafailov et al. (2023)) simplified RLHF by eliminating the reward model. Rafailov et al. (2024) analyze scaling properties and show the reference policy implicitly weights preferences by volume. Kirk et al. (2024) study how DPO's weighting affects fairness across demographic groups. Zhao et al. (2024) propose fairness-constrained variants of DPO.

AI Review Assistants and Peer Review Fairness: The paper-reviewer matching problem and its fairness implications are studied by Kobren, Saha, and McCallum (2019) (maximizing paper-reviewer compatibility) and Stelmakh, Shah, and Singh (2021) (fairness constraints in assignment). Shah (2022) analyze bias in peer review systems. AI-assisted peer review is explored by Liang et al. (2023), though fairness issues remain underexplored.

Historical Context: The questions in this chapter are not new. Arrow (1950) showed aggregation impossibilities in 1950. Thurstone (1927) introduced random utility models in 1927. Luce et al. (1959) axiomatized IIA in 1959. What's new is the **application to machine learning from human preferences at scale**—where small biases compound through automated feedback loops affecting millions of users. Classical fairness concerns now manifest in algorithmic systems with unprecedented speed and scope.

6.11 Further Reading

For readers who want to go deeper into the topics introduced in this chapter, we recommend the following:

- **Barocas, Hardt, and Narayanan (2019)**, *Fairness and Machine Learning: Limitations and Opportunities* — The comprehensive textbook on fairness in ML, covering group fairness definitions, individual fairness, impossibility results, and the sociotechnical context of algorithmic decision-making.
- **Sen (2017)**, *Collective Choice and Social Welfare* (expanded edition) — Sen's definitive treatment of social choice theory, the liberal paradox, and the distinction between revealed preferences and welfare, providing the normative foundations for this chapter.
- **Dwork et al. (2012)**, "Fairness Through Awareness" — The foundational paper on individual fairness, formalizing the Lipschitz condition that similar individuals should receive similar outcomes and introducing the metric learning challenge for fairness.
- **Fürnkranz and Hüllermeier (2010)**, *Preference Learning* — A textbook covering preference learning from a machine learning perspective, including label ranking, instance ranking, and object ranking, complementing this book's focus on human preference data.
- **Casper et al. (2023)**, "Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback" — A systematic survey of RLHF failure modes, including reward hacking, distributional shift, and the challenges of aligning AI systems with diverse human values.
- **Kleinberg, Lakkaraju, et al. (2018)**, "Human Decisions and Machine Predictions" — Formalizes the gap between behavior and mental state (the inversion problem) in the context of judicial bail decisions, showing how observed choices can systematically diverge from underlying preferences.

– **Hashimoto et al. (2018)**, "Fairness Without Demographics in Repeated Loss Minimization" — Introduces distributionally robust optimization for fairness, showing how to protect minority groups without requiring explicit demographic labels, relevant to the compounding unfairness analysis in this chapter.

This chapter synthesizes insights across these literatures to provide integrated guidance for building fairer preference learning systems.

References

Ariely, Dan. 2008. "Predictably Irrational: The Hidden Forces That Shape Our Decisions."

Arrow, Kenneth J. 1950. "A Difficulty in the Concept of Social Welfare." *Journal of Political Economy* 58 (4): 328–46.

Barocas, Solon, Moritz Hardt, and Arvind Narayanan. 2019. *Fairness and Machine Learning: Limitations and Opportunities*. fairmlbook.org.

Casper, Stephen et al. 2023. "Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback." *arXiv Preprint arXiv:2307.15217*.

Chen, Yuntao et al. 2024. "Compounding Geometric Errors in Learning to Correct." *arXiv Preprint*.

Debreu, Gerard. 1960. "Review of r. D. Luce, Individual Choice Behavior." *American Economic Review* 50: 186–88.

Dwork, Cynthia, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2012. "Fairness Through Awareness." In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12)*, 214–26. ACM. https://doi.org/10.1145/2090236.2090255.

Ensign, Danielle, Sorelle A. Friedler, Scott Neville, Carlos Scheidegger, and Suresh Venkatasubramanian. 2018. "Runaway Feedback Loops in Predictive Policing." *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, 160–71.

Friedler, Sorelle A., Carlos Scheidegger, and Suresh Venkatasubramanian. 2021. "The (Im)possibility of Fairness: Different Value Systems Require Different Mechanisms for Fair Decision Making." *Communications of the ACM* 64 (4): 136–43.

Fürnkranz, Johannes, and Eyke Hüllermeier. 2010. *Preference Learning*. Springer.

Ganguli, Deep et al. 2023. "The Capacity for Moral Self-Correction in Large Language Models." *arXiv Preprint arXiv:2302.07459*.

Green, Ben, and Yiling Chen. 2019. "The Principles and Limits of Algorithm-in-the-Loop Decision Making." *Proceedings of the ACM on Human-Computer Interaction* 3 (CSCW): 1–24.

Hardt, Moritz, Eric Price, and Nathan Srebro. 2016. "Equality of Opportunity in Supervised Learning." In *Advances in Neural Information Processing Systems*, 29:3323–31.

Hashimoto, Tatsunori, Megha Srivastava, Hongseok Namkoong, and Percy Liang. 2018. "Fairness Without Demographics in Repeated Loss Minimization." *Proceedings of the 35th International Conference on Machine Learning*, 1929–38.

Kirk, Hannah Rose et al. 2023. "Personalisation Within Bounds: A Risk Taxonomy and Policy Framework for the Alignment of Large Language Models with Personalised Feedback." *arXiv Preprint arXiv:2303.05453*.

——— et al. 2024. "Understanding the Effects of RLHF on LLM Generalisation and Diversity." *arXiv Preprint arXiv:2310.06452*.

Kleinberg, Jon, Himabindu Lakkaraju, Jure Leskovec, Jens Ludwig, and Sendhil Mullainathan. 2018. "Human Decisions and Machine Predictions." *The Quarterly Journal of Economics* 133 (1): 237–93. https://doi.org/10.1093/qje/qjx032.

Kleinberg, Jon, Jens Ludwig, Sendhil Mullainathan, and Cass R. Sunstein. 2018. "Algorithmic Fairness." *AEA Papers and Proceedings* 108: 22–27.

Kobren, Ari, Barna Saha, and Andrew McCallum. 2019. "Paper Matching with Local Fairness Constraints." *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1247–57.

Liang, Weixin et al. 2023. "Rethinking the Reviewer Assignment Problem."
Liu, Lydia T., Sarah Dean, Esther Rolf, Max Simchowitz, and Moritz Hardt. 2018. "Delayed Impact of Fair Machine Learning." *Proceedings of the 35th International Conference on Machine Learning*, 3150–58.
Luce, R Duncan et al. 1959. *Individual Choice Behavior*. Vol. 4. Wiley New York.
Mas-Colell, Andreu, Michael Dennis Whinston, Jerry R Green, et al. 1995. *Microeconomic Theory*. Vol. 1. Oxford university press New York.
Maystre, Lucas, and Matthias Grossglauser. 2015. "Fast and Accurate Inference of Plackett-Luce Models." *Advances in Neural Information Processing Systems* 28.
McFadden, Daniel. 1974. "Conditional Logit Analysis of Qualitative Choice Behavior." *Frontiers in Econometrics*, 105–42.
———. 1978. "Modelling the Choice of Residential Location." *Transportation Research Record* 673: 72–77.
———. 2000. "Disaggregate Behavioral Travel Demand's RUM Side: A 30-Year Retrospective." *Travel Behaviour Research: The Leading Edge*, 17–63.
Mullainathan, Sendhil et al. 2022. "Machine Learning and the Underpricing of High-Dimensional Information: Evidence from Patents."
Nisan, Noam, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. 2007. *Algorithmic Game Theory*. Cambridge University Press.
Nissenbaum, Helen. 2009. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press.
Pattanaik, Prasanta K. 1978. "Strategy and Group Choice."
Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." https://arxiv.org/abs/2305.18290.
———. 2024. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." *Advances in Neural Information Processing Systems* 36: 53728–41.
Rakova, Bogdana, Jingying Yang, Henriette Cramer, and Rumman Chowdhury. 2021. "Where Responsible AI Meets Reality: Practitioner Perspectives on Enablers for Shifting Organizational Practices." *Proceedings of the ACM on Human-Computer Interaction* 5 (CSCW1): 1–23.
Sah, Saumya et al. 2024. "Whose Preferences Should AI Align With?"
Schuler, Douglas, and Aki Namioka. 1993. "Participatory Design: Principles and Practices."
Sen, Amartya. 1970. "The Impossibility of a Paretian Liberal." *Journal of Political Economy* 78 (1): 152–57. https://doi.org/10.1086/259614.
———. 1977. "Rational Fools: A Critique of the Behavioral Foundations of Economic Theory." *Philosophy & Public Affairs* 6 (4): 317–44.
———. 1983. "Liberty and Social Choice." *The Journal of Philosophy* 80 (1): 5–28.
———. 2017. *Collective Choice and Social Welfare: An Expanded Edition*. Harvard University Press.
Shah, Nihar B. 2022. "Designing Fair Peer Review Systems." *arXiv Preprint arXiv:2201.13246*.
Sloane, Mona, Emanuel Moss, Olaitan Awomolo, and Laura Forlano. 2020. "Participation Is Not a Design Fix for Machine Learning." *arXiv Preprint arXiv:2007.02423*.
Stelmakh, Ivan, Nihar B. Shah, and Aarti Singh. 2021. "PeerReview4All: Fair and Accurate Reviewer Assignment in Peer Review." *Journal of Machine Learning Research* 22 (1): 163:1–66.
Thurstone, Louis Leon. 1927. "A Law of Comparative Judgment." *Psychological Review* 34 (4): 273–86.
Train, Kenneth E. 2009. *Discrete Choice Methods with Simulation*. Cambridge university press.
Tschantz, Michael Carl. 2020. "Contextual Integrity Principles for Fairness in Learning."
Zhao, Sicheng et al. 2024. "Fair RLHF: Integrating Fairness in Reinforcement Learning from Human Feedback."

7 Conclusion

Machine learning systems are increasingly shaping consequential decisions in our lives—from the content we consume to the opportunities we receive. A central challenge in developing trustworthy AI is ensuring these systems align with human preferences rather than optimizing for proxies that diverge from what people actually value. This book has presented a comprehensive framework for learning from human feedback, drawing on nearly a century of research across economics, psychology, statistics, and computer science.

7.1 Our Approach

This book has taken an interdisciplinary approach to machine learning from human preferences, weaving together insights from diverse fields into a unified technical and conceptual framework.

7.1.1 Foundations from Multiple Disciplines

Our treatment began with foundational models that originated outside of machine learning. The Bradley-Terry model (), introduced in 1952 for ranking chess players, provides the mathematical backbone for modern preference learning—including state-of-the-art methods like Direct Preference Optimization for language model alignment. Thurstone's law of comparative judgment from psychophysics (1927), Luce's choice axiom (1959), and McFadden's discrete choice econometrics (1974) all inform how we model the relationship between latent utilities and observed choices.

This interdisciplinary foundation is not merely historical context. Understanding *why* the Bradley-Terry model arises from random utility assumptions, and *when* its key assumption—Independence of Irrelevant Alternatives—fails, equips practitioners to recognize the limitations of their methods. The red-bus/blue-bus problem and preference heterogeneity are not abstract concerns but practical challenges that arise in real applications.

7.1.2 From Axioms to Algorithms

The book progressed from axiomatic foundations to practical algorithms:

Chapter 1 established the mathematical language of preferences. We showed how random utility models with Gumbel-distributed noise yield the tractable logit form, and how the IIA axiom collapses an exponentially large preference space to a linear number of parameters. These

are not arbitrary modeling choices—they encode specific assumptions about human cognition that may or may not hold in practice.

Chapter 2 developed estimation methods for preference models. Maximum likelihood estimation provides point estimates efficiently, while Bayesian inference quantifies uncertainty—essential for downstream decision-making. The Elo rating system, used from chess to video games, emerges naturally as stochastic gradient descent on the Bradley-Terry likelihood. Gaussian Processes offer flexible nonparametric alternatives when linear reward functions are too restrictive.

Chapter 3 addressed active data collection. Human feedback is expensive; we cannot afford to collect data naively. Fisher information quantifies how much each comparison teaches us about preferences, enabling intelligent query selection. The connection to optimal experimental design—A-optimal, D-optimal, E-optimal criteria—grounds active learning in classical statistical theory.

Chapter 4 turned to decision-making under learned preferences. Thompson Sampling elegantly balances exploration and exploitation by sampling from the posterior and acting optimally under the sample. Dueling bandits extend multi-armed bandits to the comparison setting, with distinct winner concepts (Condorcet, Borda, von Neumann) and regret notions appropriate for different applications. Preferential Bayesian Optimization unified Gaussian Process preference models from earlier chapters with sequential decision-making, enabling optimization of complex objectives from comparison feedback alone. The chapter culminated in the Cooperative Inverse Reinforcement Learning (CIRL) framework, which reconceived preference learning as a cooperative game between human and agent—shifting from passive data collection to interactive collaboration where the agent's queries themselves shape the human's ability to communicate preferences.

Chapter 5 examined aggregation when multiple stakeholders have heterogeneous preferences. Arrow's Impossibility Theorem and the Gibbard-Satterthwaite Theorem establish fundamental limits on what any aggregation mechanism can achieve—yet domain restrictions like single-peaked preferences offer practical escape routes. A key result connected the Borda count to Direct Preference Optimization, showing that DPO finds the policy that wins the most head-to-head matchups ()—giving modern alignment methods a precise social choice interpretation. Sen's liberal paradox () revealed the tension between respecting individual rights and achieving Pareto efficiency when preferences are "nosy," while the inversion problem () cautioned that observed behavior may not reflect true preferences. The Community Notes case study () illustrated how these ideas play out in a deployed system. Mechanism design, including VCG auctions, showed how to align individual incentives with collective objectives—but at a cost.

7.1.3 The Critical Turn

A distinctive feature of this book is **Chapter 6**, which stepped back from technical methods to ask a normative question: *Whose preferences matter?*

Every technical choice in the preference learning pipeline—from who gets queried to how preferences are aggregated—embeds value judgments about whose interests count. The Fisher information machinery from Chapter 3 maximizes informativeness per query, but information-maximizing strategies can systematically underquery minority populations whose preferences are "easier" to model. Assuming IIA treats context-dependent preferences as irrational, potentially disadvantaging overburdened users whose choices are shaped by external constraints rather than stable preferences. Using existing data to define a reference policy encodes historical inequities into the learning objective.

This critical lens does not reject the technical methods developed in earlier chapters. Rather, it calls for conscious recognition that preference learning is not value-neutral. The designer's choices shape who benefits and who is harmed. Fairness constraints, stratified evaluation, and participatory design are not add-ons but essential components of responsible system development.

7.2 Lessons from the Alignment Era

The rapid adoption of RLHF and DPO for language model alignment—beginning with InstructGPT in 2022 and accelerating through ChatGPT, Claude, and their successors—has stress-tested preference learning at a scale that the field's founders could not have anticipated. Several lessons have emerged from this experience that inform how we should think about the methods developed in this book.

7.2.1 What Worked

Bradley-Terry is remarkably effective in practice. Despite its strong assumptions (IIA, homogeneous annotators, context-independence), the Bradley-Terry model has proven to be a workhorse for reward modeling and DPO. The simplicity that makes it analytically tractable—a single parameter per item, sigmoid link function, logistic likelihood—also makes it computationally efficient at scale. The DPO reformulation (Chapter 1,) eliminates the separate reward model entirely, showing that the Bradley-Terry likelihood can be directly optimized as a policy objective. This is a striking vindication of the classical models that form the backbone of this book.

Active learning reduces annotation cost. The Fisher information framework from Chapter 3 has proven practically valuable for selecting which comparisons to annotate. ADPO () and related methods demonstrate that intelligent query selection can reduce annotation budgets by 30-50% compared to random sampling, a substantial saving when human preference labels cost \$1-10 each and training runs require tens of thousands of comparisons.

Uncertainty quantification matters for deployment. The Bayesian methods from Chapter 2 and the posterior-based decision-making from Chapter 4 (Thompson Sampling, PBO) are not merely academic exercises. In production systems, knowing *how confident* the model is in its preference predictions enables calibrated decisions about when to deploy new model versions, when to collect more data, and when to fall back to conservative defaults.

7.2.2 What Surprised Us

The gap between reward models and human judgment is smaller than expected—but the remaining gap is consequential. Modern reward models trained via Bradley-Terry achieve high agreement with held-out human preferences (often 70-80% accuracy on binary comparisons). But the errors are not random: they cluster around responses that are subtly harmful, sycophantic, or superficially impressive but factually wrong. These failure modes are precisely the cases where getting preferences right matters most, and they suggest that the IIA assumption—which treats all comparisons as equally well-modeled—may need to be relaxed for safety-critical applications.

Goodhart's Law is real and pernicious. When models are optimized against a learned reward function, they inevitably find ways to achieve high reward without satisfying the underlying human preferences. This "reward hacking" is a direct consequence of the gap between the proxy (learned reward) and the target (true human preferences). The phenomenon manifests as models that produce longer responses (because annotators weakly prefer length), use confident-sounding language regardless of accuracy, or agree with the user's stated views rather than providing honest assessments. Recognizing reward hacking as an instance of Goodhart's Law—"when a measure becomes a target, it ceases to be a good measure"—connects it to a rich literature in economics and public policy.

Annotator disagreement is signal, not noise. Early RLHF work treated disagreement among annotators as measurement error to be averaged away. But Chapter 5's analysis of preference aggregation suggests a different interpretation: annotators disagree because they genuinely hold different preferences, not (only) because they make mistakes. The DPO-Borda connection () shows that aggregating via majority vote implicitly selects the Borda winner—which may not be the socially optimal choice when minority preferences carry important information. This has led to growing interest in pluralistic alignment approaches that maintain multiple preference models rather than collapsing to a single aggregate.

7.2.3 What Broke

The inversion problem at scale. Chapter 5's discussion of the inversion problem ()—that observed behavior does not equal underlying preferences—takes on new urgency at the scale of commercial annotation. Annotators working under time pressure, fatigue, and piece-rate compensation produce preference labels that reflect their working conditions as much as their genuine judgments. Studies have documented systematic biases: annotators favor shorter responses when fatigued, prefer responses that match their cultural background, and exhibit position bias (favoring the first response presented). These are precisely the distortions that Chapter 6's fairness analysis predicts—and they propagate through the entire pipeline.

IIA violations in practice. The Independence of Irrelevant Alternatives assumption, whose limitations we analyzed theoretically in Chapter 1 (the red-bus/blue-bus problem), manifests concretely in LLM alignment. Annotators' preference between two responses depends on what other responses they have recently evaluated (contrast effects), how the comparison is framed (whether they are asked "which is better" versus "which is less harmful"), and even

the order of presentation. These violations suggest that richer preference models—context-dependent Bradley-Terry, mixed logit, or the random-coefficients models from Chapter 1—may be needed for high-stakes applications.

Single-turn preferences do not capture long-horizon value. Most preference learning for language models collects feedback on individual responses in isolation. But users ultimately care about the quality of extended interactions—whether the model is consistently helpful over a conversation, whether it remembers context appropriately, and whether it knows when to ask clarifying questions versus provide direct answers. Extending preference models from single comparisons to trajectory-level feedback remains an open challenge that connects to the sequential decision-making framework of Chapter 4.

7.3 Forwarding the Field

Machine learning from human preferences is a vibrant area of research with many open challenges. We highlight several directions that we believe are particularly important for the field's advancement.

7.3.1 Beyond Pairwise Comparisons

Much of this book focused on pairwise comparisons—the simplest and most well-understood form of preference data. But humans express preferences in many ways:

- **Natural language feedback**: "This response is helpful but could be more concise" conveys richer information than a binary comparison. Recent work on learning from critiques and natural language rewards suggests that language feedback can be formalized as soft constraints on the reward function, though the mapping from language to reward remains noisy and context-dependent.
- **Partial rankings**: A user might confidently rank their top three choices but be uncertain about the rest. Plackett-Luce models () handle full rankings, but principled treatment of partial and uncertain rankings requires extensions that allow "I don't know" or "these are roughly equivalent" responses.
- **Ratings and Likert scales**: Absolute judgments on a scale provide cardinal (not just ordinal) information, though they introduce calibration challenges across annotators—one person's 4 out of 5 may be another's 3. Anchoring and cross-annotator calibration methods are active areas of research.
- **Implicit signals**: Engagement time, scroll patterns, and click behavior reveal preferences without explicit queries. But these signals are heavily confounded by interface design, attention patterns, and the inversion problem (Chapter 5)—longer engagement may reflect confusion rather than interest.

Integrating these diverse modalities into unified preference models is an important open problem. Each modality has strengths: pairwise comparisons are reliable but expensive; implicit

signals are cheap but noisy; natural language is rich but difficult to formalize. Hybrid approaches that combine multiple feedback types, weighting each appropriately, hold promise for more efficient and accurate preference learning.

7.3.2 Scalable Oversight

As AI systems become more capable, the tasks they perform become harder for humans to evaluate. A language model writing code may produce solutions that work but are subtly flawed in ways a non-expert cannot detect. A scientific assistant might propose experiments whose validity requires deep domain expertise to assess.

Scalable oversight addresses this challenge through techniques like:

- **Recursive reward modeling**: Decomposing complex evaluation tasks into simpler subtasks that humans can reliably judge. The key insight is that while evaluating a complete solution may be hard, verifying individual steps is often feasible. This connects to the preference elicitation framework of Chapter 3: by decomposing evaluation, we can apply Fisher information analysis to each subtask and select the most informative verification queries.
- **AI-assisted evaluation**: Using AI systems to help humans provide more accurate feedback, while remaining vigilant about circular dependencies. RLAIF (reinforcement learning from AI feedback) replaces human annotators with AI evaluators, raising questions about whether the preference signal is genuinely informative or merely reflects the evaluator model's own biases.
- **Debate and amplification**: Adversarial procedures where AI systems critique each other's outputs, surfacing flaws that a single evaluator might miss. This can be formalized as a zero-sum game, connecting to the von Neumann winner concept from Chapter 4's dueling bandits framework.

The CIRL framework from Chapter 4 models the cooperative structure of oversight, where the human's ability to evaluate depends on how the agent presents its outputs. But scaling these ideas to superhuman AI systems—where the agent's capabilities exceed the evaluator's—remains one of the most important open frontiers in AI safety.

7.3.3 Temporal Dynamics and Preference Change

This book has largely treated preferences as static: a user has fixed utilities V_j that we aim to estimate. In reality, preferences evolve over time. A user's taste in music changes as they discover new genres; a society's norms around acceptable content shift across decades; an annotator's judgment drifts as they gain experience with the task.

Modeling temporal preference dynamics introduces several challenges:

- **Distinguishing drift from noise**: Is a changed preference a genuine evolution (the user's taste matured) or a noisy observation (the user was tired that day)? The Bayesian framework from Chapter 2 can be extended with time-varying priors—for example, a random walk on utilities—but choosing the right rate of change requires domain knowledge.
- **Non-stationarity in online learning**: The Elo rating system (Chapter 2,) implicitly handles non-stationarity through its constant step size K, which weights recent observations more heavily. But the optimal K depends on the rate of change, creating a tradeoff between responsiveness and stability.
- **Retroactive fairness**: If a system learns preferences from historical data that reflects outdated norms, deploying those preferences perpetuates past values. Chapter 6's pipeline framework ($\mathcal{E} \to \mathcal{L} \to \mathcal{A} \to \mathcal{D}$) provides a lens for auditing where temporal biases enter.

7.3.4 Heterogeneity and Personalization

This book has established that preference heterogeneity is not a nuisance but a fundamental feature of human populations. Chapter 1 introduced mixture models and K-dimensional factor models to capture latent preference structure. Chapter 5's analysis of the inversion problem () showed that observed behavior can systematically misrepresent preferences for overburdened groups, and the liberal paradox () revealed when respecting diverse preferences conflicts with collective efficiency. Chapter 6 demonstrated how these issues compound through feedback loops across the preference learning pipeline.

Yet significant open problems remain:

- **Scalable mixture models** that identify latent preference clusters in high-dimensional settings without requiring users to self-identify. Variational inference and amortized methods show promise for scaling mixture-of-experts preference models to millions of users.
- **Contextual preference models** that account for how the same user's preferences vary with cognitive load, time pressure, and framing—building on the contextual Bradley-Terry models introduced in Chapter 6. The challenge is distinguishing genuine context-dependence from IIA violations.
- **Personalization with fairness constraints** that provide tailored assistance while ensuring no group is systematically underserved. Multi-objective optimization frameworks that Pareto-balance personalization and equity are an active area of research.
- **Cold-start and few-shot preference learning**: How should a system behave for a new user with no preference history? Transfer learning from population-level models, combined with the active elicitation strategies from Chapter 3, can accelerate personalization while managing uncertainty.

The tension between personalization (giving each user what they want) and fairness (ensuring equitable outcomes across groups) is not fully resolved by the technical tools we have today. Progress requires both better algorithms and clearer normative frameworks for navigating these tradeoffs.

7.3.5 *Fairness and Value Alignment*

Chapter 6 introduced fairness considerations; making them operational requires ongoing research:

- **Fairness metrics for preference learning**: Standard fairness definitions (demographic parity, equalized odds) were developed for classification. What are the appropriate analogs for preference models? When is it fair for a system to learn different preference models for different groups? The pipeline framework from Chapter 6 suggests that fairness must be assessed at each stage ($\mathcal{E}, \mathcal{L}, \mathcal{A}, \mathcal{D}$), not just at the final output.
- **Value alignment under disagreement**: When stakeholders have genuinely conflicting preferences, what should a system optimize for? Majority preferences? Pareto improvements? Maximin welfare? The social choice results of Chapter 5—particularly Arrow's theorem—tell us that no aggregation rule is universally satisfactory, but they also point toward domain restrictions and mechanism design as paths forward.
- **Transparency and contestability**: Users should understand why a system makes the recommendations it does, and have meaningful recourse when they disagree. This requires preference models that are not only accurate but interpretable—a challenge for the high-dimensional factor models and GP-based approaches developed in earlier chapters.
- **Pluralistic alignment**: Rather than learning a single aggregate preference model, maintain multiple models reflecting different value systems and allow users or institutions to choose among them. This connects to the mixture models of Chapter 1 and the multi-issue voting framework of Chapter 5.

These are not purely technical questions—they require engagement with philosophy, law, and democratic theory. But technical researchers must be part of the conversation, ensuring that fairness frameworks are implementable and that their limitations are well understood.

7.3.6 *Foundation Model Alignment*

Large language models trained via RLHF and DPO represent the most prominent current application of preference learning. Yet significant challenges remain:

- **Reward hacking**: Models may find ways to achieve high reward without actually satisfying human preferences—exploiting ambiguities in the reward signal rather than genuinely helpful behavior. This is a manifestation of Goodhart's Law: when the learned reward becomes the optimization target, it ceases to be a faithful proxy for human preferences. Mitigations include reward model ensembles, KL-constrained optimization (as in DPO's implicit KL penalty), and regular recalibration against fresh human judgments.
- **Distributional shift**: Preferences collected in one context may not generalize to deployment settings, especially as models are used for novel tasks. The Bradley-Terry model assumes a fixed set of items with stable utilities, but in practice the "items" (model responses) change as the model improves, creating a moving target that requires online adaptation.

- **Constitutional AI and rule-based approaches**: Can we move beyond learning from examples to learning from principles? Constitutional AI proposes specifying desirable behavior through written rules rather than example comparisons, raising questions about how to reconcile principle-based and preference-based approaches. The connection to single-peaked preferences (Chapter 5) is suggestive: if principles define a partial order over responses, they may restrict the preference domain enough to escape aggregation impossibilities.
- **Multi-objective alignment**: Real-world alignment requires balancing multiple objectives simultaneously—helpfulness, harmlessness, honesty, and more. This is fundamentally a multi-objective optimization problem where different objectives can conflict, connecting to the social choice framework of Chapter 5.

The alignment of foundation models with human values is arguably the most consequential application of preference learning today. Progress in this area requires both fundamental research on preference modeling and careful empirical work on what actually makes language model outputs helpful, harmless, and honest.

7.4 Capstone Projects

The chapters of this book develop a rich toolkit—from preference models and estimation to active elicitation, sequential decisions, social aggregation, and fairness—but the deepest understanding comes from applying these ideas to open-ended problems. The following capstone projects are designed as extended investigations suitable for individual or small-team work. Each project is inspired by the format of the CS329H course project: students produce a pre-analysis plan, a final manuscript in NeurIPS format (up to 8 pages), and reproducible code. Projects span the book's six main chapters and range from focused empirical studies to more ambitious research contributions. Difficulty is indicated by stars: * (one person, roughly four weeks), ** (one to two people, roughly six weeks), and *** (two to three people, eight or more weeks).

7.4.1 Project 1: When Does IIA Fail? Detecting and Modeling Context Effects in Preference Data (*)

Description. The Independence of Irrelevant Alternatives assumption () is the backbone of Bradley-Terry and Plackett-Luce models, yet it is routinely violated in practice—the "red-bus/blue-bus" problem being the canonical example. This project investigates *when and how much* IIA violations matter by designing controlled experiments (or analyzing existing preference datasets) to detect context effects, then comparing Bradley-Terry against richer models that relax IIA.

Suggested methods.

- Fit a standard Bradley-Terry model () and a mixed logit or nested logit model () to the same preference data.

- Use likelihood ratio tests or AIC/BIC () to quantify whether the richer model provides a statistically significant improvement.
- Construct synthetic "red-bus/blue-bus" scenarios by introducing near-duplicate items into a choice set and measuring the distortion in estimated utilities.

Expected deliverables. A manuscript reporting (1) a dataset with documented IIA violations, (2) a quantitative comparison of Bradley-Terry vs. at least one IIA-relaxing model, and (3) practical guidelines for when practitioners should move beyond Bradley-Terry. Code for all experiments with reproducible results.

7.4.2 Project 2: Bayesian vs. Frequentist Preference Learning on Real Annotation Data (*)

Description. Chapter 2 presents three estimation paradigms—maximum likelihood (), Bayesian inference via MCMC (), and online learning via the Elo system (). How do these approaches compare on real-world human preference data, especially in low-data regimes where uncertainty quantification matters most? This project conducts a systematic comparison using publicly available LLM preference datasets.

Suggested methods.

- Implement MLE with L2 regularization (), Bayesian inference with the Laplace approximation (), and Elo updates () for the same Bradley-Terry model.
- Evaluate predictive accuracy (AUC), calibration, and posterior coverage as a function of dataset size by subsampling.
- Assess computational cost and convergence diagnostics ().

Expected deliverables. A manuscript with learning curves showing accuracy and calibration vs. sample size for each method, a discussion of when Bayesian uncertainty is worth the computational overhead, and fully documented code including data preprocessing scripts.

7.4.3 Project 3: Active Preference Elicitation for Personalized Recommendation (**)

Description. When human feedback is expensive, active query selection can dramatically reduce annotation cost. This project builds an end-to-end active preference elicitation system for a concrete domain—such as food preferences, music taste, or movie recommendations—and measures how much the Fisher information framework improves over random querying.

Suggested methods.

- Implement a Rasch or factor model () with Fisher-information-based query selection (,).
- Compare A-optimal, D-optimal, and E-optimal criteria () for selecting the next comparison to present.
- Collect real preference data from at least 10 participants (or use a realistic simulator) and measure the reduction in queries needed to reach a target prediction accuracy.

– Optionally extend to the pairwise setting using the active pair selection framework from .

Expected deliverables. A working interactive system (command-line or web-based) that adaptively selects queries, an empirical evaluation showing query savings relative to random selection, and a manuscript discussing the practical benefits and limitations of active elicitation.

7.4.4 Project 4: Dueling Bandits for A/B Testing with Preference Feedback (**)

Description. Traditional A/B testing compares treatments using scalar metrics (click-through rate, revenue), but in many settings the outcome of interest is a human preference judgment—e.g., "Which UI layout do you prefer?" This project formulates A/B testing as a dueling bandit problem and evaluates whether dueling bandit algorithms can identify the best variant faster than naive round-robin comparisons.

Suggested methods.

– Implement at least two dueling bandit algorithms from Chapter 4, such as RUCB or Double Thompson Sampling, targeting different winner concepts (Condorcet vs. Borda,).
– Define appropriate regret metrics for the A/B testing setting and track cumulative regret over time.
– Simulate a realistic A/B testing scenario with 5–10 variants and heterogeneous user preferences (using mixture models from to generate synthetic users).
– Compare sample efficiency against uniform random pairing and standard Thompson Sampling () on scalar proxy metrics.

Expected deliverables. A manuscript presenting regret curves and sample complexity comparisons, a discussion of which winner concept is most appropriate for A/B testing, and modular code that could be adapted to other dueling bandit applications.

7.4.5 Project 5: Preferential Bayesian Optimization for Human-in-the-Loop Design (**)

Description. Preferential Bayesian Optimization (PBO) enables optimization of complex objectives—such as the taste of a recipe, the aesthetics of a generated image, or the comfort of a robot trajectory—from pairwise human feedback alone. This project applies PBO to a concrete design problem where scalar evaluation is difficult but pairwise comparison is natural.

Suggested methods.

– Implement the GP-based PBO framework from Chapter 4, including a Gaussian Process preference model with Laplace approximation () and an acquisition function (Expected Improvement of the Copeland score or Thompson Sampling).
– Choose a design domain with a continuous parameter space (e.g., color palettes, font configurations, audio equalization settings, or procedurally generated environments).

- Collect preference feedback from at least 5 participants and track convergence to their preferred design.
- Compare PBO against random search and, if a scalar proxy is available, against standard Bayesian Optimization.

Expected deliverables. A manuscript reporting convergence rates and user satisfaction, a working PBO system with a human-facing interface, and an analysis of how posterior uncertainty evolves with the number of comparisons.

7.4.6 *Project 6: The DPO-Borda Connection: Empirical Validation on Multi-Annotator Data (**)*

Description. Chapter 5 establishes a theoretical connection between Direct Preference Optimization and the Borda count (): DPO finds the policy that maximizes pairwise win rate, i.e., the Borda winner. But does this hold empirically when annotators are heterogeneous and preferences are noisy? This project tests the DPO-Borda connection on real multi-annotator preference data and investigates when the Borda winner diverges from other social choice solutions.

Suggested methods.

- Use a publicly available multi-annotator preference dataset (e.g., from Chatbot Arena or similar).
- Compute the Borda ranking, Condorcet ranking (if it exists), and plurality ranking from raw annotations.
- Train a DPO model and compare its implicit ranking to each social choice solution.
- Analyze cases where the rankings diverge: are these cases where annotators have genuinely heterogeneous preferences, or where the data is simply noisy? Use the mixture model framework from to test for latent preference clusters.

Expected deliverables. A manuscript quantifying the agreement between DPO-implied rankings and classical social choice rankings, an analysis of when and why they diverge, and reproducible code for all experiments.

7.4.7 *Project 7: Auditing a Preference Learning Pipeline for Compounding Bias (**)*

Description. Chapter 6 demonstrates that small biases at each stage of the preference learning pipeline—elicitation, learning, aggregation, decision—can compound into large disparities (). This project operationalizes that insight by constructing a complete pipeline simulation, injecting realistic biases at each stage, and measuring how they interact.

Suggested methods.

- Build a simulated preference learning pipeline following the four-stage framework ($\mathcal{E} \to \mathcal{L} \to \mathcal{A} \to \mathcal{D}$) from .

- Model two or more user groups with different base rates of participation, different noise levels, and different preference distributions.
- At each pipeline stage, introduce a specific bias mechanism: (1) undersampling a group in elicitation (), (2) IIA misspecification for context-dependent preferences (), (3) volume-weighted aggregation (), and (4) liberal vs. illiberal decision-making ().
- Measure group-level disparities in recommendation quality at the pipeline output, and decompose total disparity into contributions from each stage.

Expected deliverables. A manuscript with a bias decomposition showing how much each pipeline stage contributes to overall unfairness, concrete recommendations for which stage to prioritize when mitigating bias, and a reusable simulation framework.

7.4.8 Project 8: Mechanism Design for Incentive-Compatible Preference Elicitation (***)

Description. When users have strategic incentives—e.g., inflating ratings to influence recommendations, or misreporting preferences in a voting system—naive preference elicitation produces biased data. This project designs and evaluates an incentive-compatible mechanism for a specific preference learning application, drawing on the mechanism design framework from Chapter 5 ().

Suggested methods.

- Choose a concrete setting where strategic misreporting is plausible (e.g., peer review, course evaluation, or collaborative filtering).
- Design a mechanism that incentivizes truthful reporting, building on VCG mechanisms or peer prediction from .
- Prove or empirically demonstrate the mechanism's incentive compatibility properties.
- Simulate strategic agents using a mixture of truthful and strategic types, and compare preference estimation quality under your mechanism vs. naive elicitation.
- Analyze the mechanism's efficiency cost (the "price of incentive compatibility") relative to a setting with truthful agents.

Expected deliverables. A manuscript with a formal mechanism description, theoretical analysis of incentive properties, simulation results comparing truthful vs. strategic settings, and implementation code. If applicable, include a discussion of how the mechanism interacts with the fairness considerations from Chapter 6 ().

7.4.9 Project 9: Cooperative Inverse Reinforcement Learning for Interactive Preference Learning (***)

Description. The CIRL framework from Chapter 4 models preference learning as a cooperative game between a human and an agent, where the agent's actions serve both to accomplish tasks and to elicit information about the human's reward function. This project implements a CIRL-style agent in a simplified domain and studies how cooperative interaction compares to passive preference observation.

Suggested methods.

- Define a gridworld or simple continuous environment where a human has a latent reward function over outcomes.
- Implement a CIRL agent that maintains a belief over reward functions and selects actions to jointly maximize expected reward and information gain, following the framework from Chapter 4.
- Compare against (1) a passive agent that observes demonstrations and infers preferences via inverse reinforcement learning, and (2) an active agent that can ask explicit comparison queries ().
- Measure convergence rate to the true reward function, cumulative regret, and the human's perceived interaction quality (if using human participants) or simulated interaction cost.

Expected deliverables. A manuscript presenting the CIRL formulation for your domain, regret comparisons across the three approaches, an analysis of when cooperative interaction provides the most benefit (e.g., ambiguous rewards, high-dimensional preference spaces), and well-documented code.

7.4.10 Project 10: Pluralistic Alignment—Learning and Serving Multiple Preference Models (***)

Description. Most preference learning systems collapse diverse human preferences into a single aggregate model. Drawing on Arrow's Impossibility Theorem (), the DPO-Borda connection (), and the fairness framework of Chapter 6, this project explores an alternative: learning *multiple* preference models that represent distinct value systems, and allowing users or institutions to choose among them.

Suggested methods.

- Start with a multi-annotator preference dataset and fit a mixture model to discover latent preference clusters (,).
- Train separate DPO or reward models for each cluster and evaluate whether cluster-specific models outperform a single aggregate model on within-cluster prediction accuracy.
- Investigate the social choice properties of the cluster-level approach: does serving cluster-specific models avoid Arrow-style impossibilities? Under what conditions does it introduce new fairness concerns (e.g., stereotype reinforcement)?
- Design a user-facing mechanism for selecting among preference models, and evaluate whether users can meaningfully choose the model that best represents their values.

Expected deliverables. A manuscript with (1) evidence that latent preference clusters exist in real data, (2) a comparison of pluralistic vs. aggregate alignment, (3) a discussion of the fairness trade-offs involved, and (4) code for cluster discovery, model training, and evaluation. This project connects nearly every chapter of the book and is suitable for an ambitious team.

7.5 A Practitioner's Checklist

For readers building preference learning systems in practice, we distill the book's lessons into a checklist of questions to address at each stage of system development.

Data Collection (Chapters 1, 3)

- What type of preference data will you collect? Pairwise comparisons, rankings, ratings, or natural language? Each carries different assumptions and failure modes.
- Who are your annotators, and how are they incentivized? Piece-rate compensation can introduce the fatigue and speed-accuracy tradeoffs that create the inversion problem (Chapter 5,).
- Are you using active query selection? If human feedback is expensive, the Fisher information framework (Chapter 3) can substantially reduce the number of comparisons needed.
- Have you considered position bias, framing effects, and other presentation artifacts that violate IIA?

Modeling (Chapters 1, 2)

- Is Bradley-Terry appropriate for your setting, or do you need richer models? If your population is heterogeneous, consider mixture models or factor models from Chapter 1.
- Are you quantifying uncertainty? Point estimates (MLE) are fast but miss the posterior information needed for downstream decision-making. The Laplace approximation provides a practical middle ground.
- How are you evaluating your model? Use multiple metrics—AUC for ranking, calibration error for probabilistic predictions, and subgroup analysis for fairness (Chapter 6).
- Is your model regularized appropriately? Too little regularization overfits to noisy preferences; too much shrinks all utilities toward zero.

Decision-Making (Chapter 4)

- Are you in a measurement setting (reduce uncertainty) or a reward maximization setting (manage uncertainty)? The choice determines whether elicitation (Chapter 3) or bandit/optimization methods (Chapter 4) are appropriate.
- If optimizing sequentially, how are you balancing exploration and exploitation? Thompson Sampling provides a principled default.
- What is your winner concept? Condorcet, Borda, and von Neumann winners can diverge—the choice encodes assumptions about what "best" means.

Aggregation and Fairness (Chapters 5, 6)

- Whose preferences are included, and whose are excluded? Convenience sampling overrepresents easily-reached populations.
- How are you aggregating across annotators? Majority vote selects the Borda winner (Chapter 5,)—is that what you want?

- Have you audited for compounding unfairness? Small biases at each pipeline stage can multiply through feedback loops (Chapter 6).
- Are your value tradeoffs explicit and documented? If not, they are implicit and unaccountable.

Deployment and Monitoring

- Are you monitoring subgroup performance over time, not just aggregate metrics?
- Do you have a plan for preference drift? User preferences and societal norms change; static models decay.
- Is there a mechanism for users to contest or correct the system's preference model?
- Have you considered the system's role as a choice architect—how the options you present shape the preferences you observe?

7.6 Scope and Limitations

This book has deliberately focused on the mathematical and algorithmic foundations of preference learning, and several important topics lie outside its scope.

Cognitive science of preference formation. We model preferences as given—either as fixed utilities or as draws from a distribution—without deeply examining how humans form preferences in the first place. The rich literature on bounded rationality, heuristics and biases, and constructed preferences suggests that human choice is more complex than any random utility model captures. Readers interested in these foundations should consult the work of Kahneman, Tversky, Slovic, and their successors.

Large-scale systems engineering. The algorithms in this book are presented at a scale suitable for understanding and experimentation. Production preference learning systems at major technology companies handle billions of data points, thousands of annotators, and models with billions of parameters. The engineering challenges of distributed training, data pipeline management, annotation quality control, and real-time serving are substantial but largely orthogonal to the algorithmic questions we address.

Legal and regulatory frameworks. The deployment of preference learning systems increasingly intersects with privacy regulation (GDPR's right to explanation, data minimization), anti-discrimination law, and sector-specific oversight (healthcare, finance, education). These legal dimensions shape what data can be collected, how it can be used, and what accountability mechanisms must be in place. We have touched on related ethical considerations in Chapter 6, but a thorough treatment of the regulatory landscape is beyond our scope.

Multi-modal and embodied preferences. This book focuses primarily on preferences over discrete items or text responses. Preferences in robotics (over trajectories), in design (over visual layouts), and in multi-modal settings (over combinations of text, images, and audio) raise additional challenges that we have only briefly touched upon.

7.7 *Final Thought*

At its core, this book has been about a simple idea: rather than specifying objectives by hand, we can learn them from human feedback. This idea is powerful because it promises AI systems that serve human values rather than proxy metrics—systems that improve as we better understand what we want.

But the idea is also subtle. Human preferences are noisy, inconsistent, context-dependent, and heterogeneous. They can be manipulated, and they evolve over time. Learning from preferences does not automatically solve the alignment problem; it transforms it into questions about whose preferences to learn from, how to aggregate them, and when to defer to them versus override them.

The technical methods in this book—Bradley-Terry models, Bayesian inference, Fisher information, Thompson Sampling, preferential Bayesian optimization, the DPO-Borda connection, CIRL, mechanism design—are tools, and they are more deeply connected than a chapter-by-chapter reading might suggest. The Elo system is stochastic gradient descent on Bradley-Terry; DPO finds the Borda winner; Fisher information drives both active elicitation and fairness auditing; CIRL reframes passive data collection as cooperative interaction. These connections form a coherent framework, not merely a collection of techniques. But like all tools, their value depends on how they are used. The same algorithms that personalize recommendations to delight users can personalize manipulation to exploit them.

We hope this book has equipped you not only with technical skills but also with the conceptual frameworks to use them wisely. The field of machine learning from human preferences is young and evolving rapidly. The researchers and practitioners who engage with it today will shape how AI systems learn from and serve humanity in the decades to come.

We invite you to contribute—to develop new methods, to apply existing ones thoughtfully, to critique approaches that fall short, and to engage with the broader societal implications of this work. The challenge of building AI systems that truly align with human values is too important to be left to any single discipline or community. It requires the collective effort of computer scientists, economists, psychologists, philosophers, policymakers, and the public.

The journey from this book's mathematical foundations to AI systems that reliably serve human flourishing is long. But every improvement in how we learn from human preferences brings us closer to that goal. We hope you will join us on this path.

8 Teaching Materials

This page collects resources for instructors using *Machine Learning from Human Preferences* in their courses. The materials below were developed for Stanford's CS329H but can be adapted to courses of varying length and focus.

8.1 Lecture Slides

Each chapter has an accompanying slide deck in RevealJS format.

Chapter	Topic	Slides
Introduction	Overview and motivation	Slides[1]
Chapter 1	Foundations	Slides[2]
Chapter 2	Learning	Slides[3]
Chapter 3	Elicitation	Slides[4]
Chapter 4	Decisions	Slides[5]
Chapter 5	Aggregation	Slides[6]
Chapter 6	Whose?	Slides[7]

8.2 Problem Sets

Problem Set	Topics
Problem Set 1[8]	Foundations
Problem Set 2[9]	Learning
Problem Set 3[10]	Elicitation
Problem Set 4[11]	Decisions

[1] ../slides/chap0.html
[2] ../slides/chap1.html
[3] ../slides/chap2.html
[4] ../slides/chap3.html
[5] ../slides/chap4.html
[6] ../slides/chap5.html
[7] ../slides/chap6.html
[8] psets/pset1.html

8.3 Suggested Course Pathways

The book supports several course configurations depending on audience and length. Chapter 1 is foundational to all pathways.

- **Applied AI** (e.g., early graduate ML course): Chapters 1, 2, and 4.
- **Discrete choice and economics**: Skim Chapter 1, then Chapters 2 and 4.
- **Deep learning focus**: Chapters 2–4.
- **Computation and society**: Chapters 1, 5, and 6.

[9] psets/pset2.html
[10] psets/pset3.html
[11] psets/pset4.html

Mathematical Notation

This appendix consolidates the mathematical notation used throughout the book. Symbols are grouped by topic; the **Introduced** column indicates the chapter where each symbol first appears.

General

Symbol	Meaning
$M \in \mathbb{N}$	Number of items (alternatives)
$j, j', k \in \{1, \dots, M\}$	Item (alternative) indices
$N \in \mathbb{N}$	Number of users (agents)
$i \in \{1, \dots, N\}$	User (agent) index
$\prec, \succ$	Weak preference relation / strict preference
$H_{ij} \in \mathbb{R}$	Latent utility of user i for item j
$V_j \in \mathbb{R}$ or $\mathbb{R}^K$	Item appeal / item embedding vector
$U_i \in \mathbb{R}^K$	User embedding / preference vector
$Y_{jj'} \in \{0, 1\}$	Binary preference outcome (1 means $j \succ j'$)
$\varepsilon_j \in \mathbb{R}$	Stochastic utility shock for item j (i.i.d.)
$\sigma(x) = 1/(1 + e^{-x})$	Logistic sigmoid function
$p(\cdot \mid \cdot)$	Conditional probability
x	Context / prompt (in LLM setting)
y, y_w, y_l	Response / winning response / losing response
$d \in \mathbb{N}$	Dimensionality of feature vectors
$x_j \in \mathbb{R}^d$	Feature vector of item j
$\mathcal{D}_t = \{(V_j, Y_{ij})\}_{j=1}^t$	Observed dataset at time t

Preference Models (Chapter 1)

Symbol	Meaning
0	Outside ("no-choice") option index
$L = (j_1, \dots, j_M)$	Full ranking (permutation of items)
$(j, \mathcal{S})$	Observation that j is chosen from $\mathcal{S}$
$H_j \in \mathbb{R}$	Latent utility (single-user case)

Symbol	Meaning
$r(x, y) \in \mathbb{R}$	Reward function
$p(j \succ k) = \sigma(V_j - V_k)$	Bradley-Terry comparison probability
$\tilde{V}_j = V_j + \epsilon_j$	Random utility (ϵ_j is noise)
$p(j \mid \mathcal{S}) = e^{V_j} / \sum_{k \in \mathcal{S}} e^{V_k}$	Multinomial logit (softmax) choice probability
$\prod_r e^{V_{j_r}} / \sum_{s \geq r} e^{V_{j_s}}$	Plackett-Luce ranking probability
$\alpha_i \in (0, 1)$	Population weight of subgroup i ($\sum_i \alpha_i = 1$)
$\beta \in \mathbb{R}^d$	Random-coefficients vector (linear RUM)
$\Sigma \in \mathbb{R}^{d \times d}$	Covariance matrix of β
$\mathcal{GP}(m, k)$	Gaussian Process with mean m and kernel k
$\ell \in \mathbb{R}^+$	Length-scale parameter (RBF kernel)
$\sigma_f^2 \in \mathbb{R}^+$	Signal variance (GP kernel)

Learning and Estimation (Chapter 2)

Symbol	Meaning
$\ell(V) = \sum \log \sigma(V_w - V_\ell)$	Bradley-Terry log-likelihood
$\hat{V}_{\text{MLE}} = \arg\max_V \ell(V)$	Maximum likelihood estimate
$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)\, p(\theta)$	Posterior distribution (Bayes' rule)
K	Elo step size / learning rate
λ	L2 regularization strength
$\pi_\theta(y \mid x)$	Policy (language model) parameterized by θ
$\pi_{\text{ref}}(y \mid x)$	Reference policy in DPO
β	DPO temperature parameter

Elicitation and Measurement (Chapter 3)

Symbol	Meaning
$\mathcal{I}(U)$	Fisher information (scalar or matrix)
Z_j	Item offset / difficulty ($-V_j$ in Rasch model)
$\Lambda = \Sigma^{-1}$	Posterior precision matrix
$\text{Rel} = 1 - \text{tr}(\hat{\Sigma}_{\text{err}})/\text{tr}(\Sigma_U)$	Reliability
$\Delta_D, \Delta_A, \Delta_E$	D-optimal, A-optimal, E-optimal acquisition functions
$a(Q)$	Acquisition function value for query Q
$H(y)$	Entropy of random variable y
$I(r; y)$	Mutual information between r and y

Sequential Decisions (Chapter 4)

Symbol	Meaning
$\tilde{U}_i^{(t)} \sim p(U_i \mid \mathcal{D}_t)$	Posterior sample (Thompson Sampling)
$\alpha(\cdot)$	Acquisition function
$R(T) = \sum_{t=1}^{T}(\mu^* - \mu_{a_t})$	Cumulative regret after T rounds
$\mu_t(\cdot),\ \sigma_t^2(\cdot)$	GP posterior mean and variance
$k(\cdot, \cdot)$	Kernel (covariance) function
$\pi_f([\mathbf{x}, \mathbf{x}'])$	Preference function: probability $\mathbf{x} \succ \mathbf{x}'$

Social Choice and Aggregation (Chapter 5)

Symbol	Meaning
$N = \{1, \dots, n\}$	Set of n voters (agents)
$A = \{a_1, \dots, a_m\}$	Set of m alternatives
$\succ_i$	Voter i's strict preference ordering
$\mathcal{L}(A)$	Set of all strict linear orders over A
$f : \mathcal{L}(A)^n \to A$	Social choice function (profile $\to$ winner)
$F : \mathcal{L}(A)^n \to \mathcal{L}(A)$	Social welfare function (profile $\to$ ranking)
SP(Y)	Single-peaked preferences on ordered set Y
peak$(\succ_i)$	Peak (ideal point) of voter i
v_i	Private valuation of agent i (mechanism design)
$\varphi(v)$	Virtual valuation (Myerson framework)

Fairness (Chapter 6)

Symbol	Meaning
$\mathcal{E}$	Elicitation policy
$\mathcal{L}$	Learning model structure
$\mathcal{A}$	Aggregation function
$\mathcal{D}$	Decision policy
$d(\cdot, \cdot)$	Distance metric for individual fairness
$\mathcal{G}$	Set of protected groups
Nosy(p)	True if preference p concerns others' choices (Sen)
$\mathcal{F}_{\text{ind}}$	Individual fairness constraint: $d(x_i, x_j) \leq \epsilon \Rightarrow d(f(x_i), f(x_j)) \leq \delta$
$\mathcal{F}_{\text{group}}$	Group fairness constraint: $\mathbb{E}[f(x) \mid G{=}g_1] = \mathbb{E}[f(x) \mid G{=}g_2]$

Symbol	Meaning
B, M, C	Behavior, Mental state, Context (inversion problem)

GLOSSARY

Key terms used throughout this book, organized alphabetically. Chapter references indicate where each concept is first introduced or primarily developed.

A-Optimal Design Experimental design that minimizes the *average* variance of parameter estimates, i.e., $\text{tr}(\mathcal{I}^{-1})$. Contrasts with D-optimal (volume) and E-optimal (worst-case). See .

Acquisition Function A function that quantifies the value of querying a particular item or pair, balancing information gain against expected reward. Used in active learning, Bayesian optimization, and dueling bandits. See .

Active Learning A paradigm where the learner adaptively selects which queries to pose (e.g., which items to compare) to maximize information gain, rather than passively receiving data. See .

Arrow's Impossibility Theorem A foundational result in social choice theory: no social welfare function over three or more alternatives can simultaneously satisfy unanimity, independence of irrelevant alternatives, and non-dictatorship. See .

Bayesian Inference A statistical framework that combines prior beliefs $p(\theta)$ with observed data via Bayes' rule to obtain a posterior distribution $p(\theta \mid \mathcal{D})$, enabling uncertainty quantification over model parameters. See .

Borda Count A positional voting rule where each voter assigns points based on rank position (highest-ranked gets $m-1$ points, next gets $m-2$, etc.). The alternative with the most total points wins. Violates IIA but satisfies other desirable properties. See .

Bradley-Terry Model A pairwise comparison model where the probability that item j beats item k is $p(j \succ k) = \sigma(V_j - V_k)$, with σ the logistic function. Equivalent to random utility with i.i.d. Gumbel noise. See .

Calibration The property that predicted probabilities match observed frequencies: if a model predicts $p(j \succ k) = 0.7$, then j should beat k approximately 70% of the time in practice. See .

Computerized Adaptive Testing (CAT) An assessment paradigm that selects test items adaptively based on a test-taker's estimated ability, using information-theoretic criteria to maximize measurement precision with fewer items. See .

Condorcet Winner An alternative that beats every other alternative in pairwise majority comparison. May not exist (see Condorcet paradox). When it exists, many argue it should be the social choice. See .

Condorcet Paradox The observation that majority preferences can cycle even when all individual preferences are transitive. With three voters ranking $A \succ B \succ C$, $B \succ C \succ A$, $C \succ A \succ B$: A beats B, B beats C, C beats A. See .

Cooperative Inverse Reinforcement Learning (CIRL) A game-theoretic framework where a robot and human jointly optimize the human's reward function, which the robot does not initially know. The robot must actively learn preferences through interaction. See Chapter 4.

Cross-Validation A model selection technique that partitions data into training and validation folds, fitting on training data and evaluating on held-out data. Used to select regularization strength, kernel hyperparameters, and model complexity. See .

D-Optimal Design Experimental design that maximizes $\det(\mathcal{J})$, the determinant of the Fisher information matrix. Geometrically, this maximally shrinks the volume of the posterior uncertainty ellipsoid. See .

Demographic Parity A group fairness criterion requiring that the probability of a positive outcome is equal across protected groups: $P(\hat{Y} = 1 \mid A = a) = P(\hat{Y} = 1 \mid A = b)$. See .

Direct Preference Optimization (DPO) A method for aligning language models directly from pairwise preference data without training a separate reward model. Equivalent to Bradley-Terry MLE where the utility is $\beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$. See .

Dueling Bandits A variant of the multi-armed bandit problem where feedback comes as pairwise comparisons ("which of these two options is better?") rather than absolute rewards. See Chapter 4.

E-Optimal Design Experimental design that maximizes $\lambda_{\min}(\mathcal{J})$, the smallest eigenvalue of the Fisher information matrix. Guards against worst-case estimation error. See .

Elo Rating System A rating system (originally for chess) that updates player ratings after each match. Mathematically equivalent to stochastic gradient ascent on the Bradley-Terry log-likelihood. See .

Equalized Odds A group fairness criterion requiring that the true positive rate and false positive rate are equal across protected groups: $P(\hat{Y} = 1 \mid Y = y, A = a) = P(\hat{Y} = 1 \mid Y = y, A = b)$ for $y \in \{0, 1\}$. See .

Exploration-Exploitation Tradeoff The fundamental tension in sequential decision-making: exploit current best knowledge for immediate reward, or explore uncertain options for future information gain. See .

Fisher Information A measure of how much information an observation carries about an unknown parameter. For the Bradley-Terry model with one comparison, $\mathcal{I}(U) = p(1-p)$ where $p = \sigma(U - V_j)$. Maximized when $p \approx 0.5$. See .

Gaussian Process (GP) A nonparametric Bayesian model that places a prior distribution over functions. In preference learning, GPs model utility functions with uncertainty, enabling principled exploration through posterior sampling. See .

Gibbard-Satterthwaite Theorem Any deterministic social choice function over three or more alternatives that is strategy-proof (no voter can benefit from misreporting preferences) and onto must be dictatorial. Implies all non-dictatorial voting rules are manipulable. See .

Goodhart's Law "When a measure becomes a target, it ceases to be a good measure." In preference learning: optimizing a learned reward model too aggressively can lead to reward hacking, where the model exploits the proxy rather than satisfying true human preferences. See Chapter 7.

Group Fairness Fairness criteria defined over protected demographic groups (e.g., requiring equal outcomes or error rates across groups). Includes demographic parity, equalized odds, and calibration. See .

Identification The property that model parameters can be uniquely recovered from observable data. In utility models, parameters are typically identified only up to location/scale transformations, requiring normalization constraints. See .

Independence of Irrelevant Alternatives (IIA) The property that the relative probability of choosing between two alternatives is unaffected by the presence or absence of other alternatives. Equivalent to i.i.d. Gumbel noise in random utility models. See .

Individual Fairness A fairness criterion requiring that similar individuals receive similar treatment: $d(f(x), f(x')) \leq L \cdot d(x, x')$ for a Lipschitz constant L and appropriate metrics. Fundamentally incompatible with group fairness in many settings. See .

Inversion Problem The challenge that preference learning pipelines can *invert* intended fairness guarantees: choices that seem fair at one stage (e.g., efficient elicitation) can produce unfair outcomes downstream. See .

Item Response Theory (IRT) A family of psychometric models relating a latent trait (ability) to observable responses. The Rasch model is a special case where $p(\text{correct}) = \sigma(U_i - V_j)$, with U_i the person's ability and V_j the item's difficulty. See .

Laplace Approximation A technique that approximates a posterior distribution with a Gaussian centered at the MAP estimate, using the Hessian of the log-posterior as the precision matrix. Essential for tractable GP-based preference models. See .

Maximum Likelihood Estimation (MLE) Point estimation by maximizing the likelihood function $\hat{\theta} = \arg\max_\theta \prod_i p(y_i \mid \theta)$. For Bradley-Terry: $\hat{V} = \arg\max \sum \log \sigma(V_w - V_\ell)$. See .

Mechanism Design The field of designing rules or institutions (mechanisms) that produce desirable outcomes even when participants act strategically in their own self-interest. Sometimes called "reverse game theory." See .

Plackett-Luce Model A ranking model that generalizes Bradley-Terry to full rankings. The probability of ranking $j_1 \succ j_2 \succ \cdots$ is $\prod_r \frac{e^{V_{j_r}}}{\sum_{s \geq r} e^{V_{j_s}}}$. Each position is chosen by softmax from the remaining alternatives. See .

Preferential Bayesian Optimization (PBO) Bayesian optimization using pairwise preference feedback instead of scalar function evaluations. Useful when humans can compare outcomes more reliably than they can assign absolute scores. See Chapter 4.

Random Utility Model (RUM) A framework where an individual's utility for item j is $\tilde{V}_j = V_j + \epsilon_j$, with V_j the deterministic component and ϵ_j random noise capturing unobserved factors. The individual chooses the item with highest realized utility. See .

Rashomon Effect The phenomenon where many substantially different models explain the data equally well. Named after the Kurosawa film. In preference learning, different utility vectors may fit identical comparison data. See .

Rasch Model A one-parameter IRT model where the probability of a correct response depends only on the difference between person ability U_i and item difficulty V_j: $p(\text{correct}) = \sigma(U_i - V_j)$. See .

Regret The cumulative difference between the reward of the optimal policy and the reward actually obtained. A standard measure of sequential decision-making performance: $R_T =$

$\sum_{t=1}^{T}[r^* - r_t]$. See .

Reinforcement Learning from Human Feedback (RLHF) A pipeline for aligning language models: (1) collect pairwise human preferences over model outputs, (2) train a reward model on these preferences, (3) optimize the language model against the reward model using RL (typically PPO). See .

Reliability In psychometrics, the proportion of observed variance attributable to true differences rather than measurement error: $\text{Rel} = 1 - \sigma^2_{\text{error}}/\sigma^2_{\text{total}}$. Ranges from 0 (pure noise) to 1 (perfect measurement). See .

Social Welfare Function A function that maps individual preference orderings to a single collective ordering over alternatives. Arrow's theorem constrains what properties such functions can simultaneously satisfy. See .

Thompson Sampling A Bayesian decision-making algorithm: sample parameters from the posterior, then act optimally given those sampled parameters. Naturally balances exploration (sampling from uncertain posteriors) and exploitation (acting optimally). See .

Thurstone Model A paired comparison model (1927) where $p(j \succ k) = \Phi\left(\frac{V_j - V_k}{\sqrt{2}\sigma}\right)$, using Gaussian noise rather than Gumbel. Practically similar to Bradley-Terry for most applications. See .

Unanimity (Pareto Efficiency) A fairness axiom: if every individual strictly prefers x to y, then the social ranking should prefer x to y. One of Arrow's three axioms. See .

Upper Confidence Bound (UCB) A decision-making strategy that selects the action with the highest optimistic estimate: $j^* = \arg\max_j[\hat{\mu}_j + \beta\hat{\sigma}_j]$, where β controls the exploration-exploitation tradeoff. See .

Index