CodeGemma: Open Code Models Based on Gemma

CodeGemma Team, Google LLC[1]

[1] See Contributions and Acknowledgments section for full author list. Please send correspondence to codegemma-team@google.com.

This paper introduces CodeGemma, a collection of specialized open code models built on top of Gemma, capable of a variety of code and natural language generation tasks. We release three model variants. CodeGemma 7B pretrained (PT) and instruction-tuned (IT) variants have remarkably resilient natural language understanding, excel in mathematical reasoning, and match code capabilities of other open models. CodeGemma 2B is a state-of-the-art code completion model designed for fast code infilling and open-ended generation in latency-sensitive settings.

Introduction

We present CodeGemma, a collection of open code models based on Google DeepMind's Gemma models (Gemma Team et al., 2024).

Continuing from Gemma pretrained models, CodeGemma models are further trained on more than 500 to 1000 billion tokens of primarily code, using the same architectures as the Gemma model family. As a result, CodeGemma models achieve state-of-the-art code performance in both completion and generation tasks, while maintaining strong understanding and reasoning skills at scale. We release a 7B code pretrained model and a 7B instruction-tuned code model. Further, we release a specialized 2B model, trained specifically for code infilling and open-ended generation. The lineage of these models is depicted in Figure 1.

We published the first collection (v1.0) on April 9th, 2024 with all three models. A month later, on May 3rd, 2024, we published a follow up (v1.1) to the pretrained 2B and instruction-tuned 7B models. Unless speed is critically important, we suggest v1.1 as it offers a well balanced quality improvement.

In this report, we provide an overview of the additions to Gemma, such as pretraining and instruction-tuning details for CodeGemma, followed by evaluations of all models across a wide variety of academic and real world tasks against similar models. Finally, we outline the areas in which CodeGemma excels and its limitations, followed by recommendations for using this model. Where applicable, we note the differences between v1.0 and v1.1.

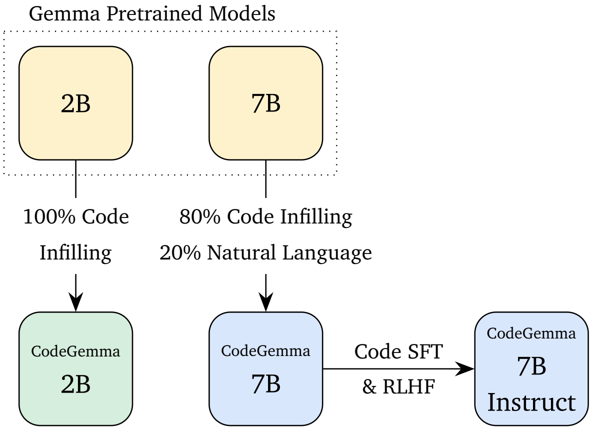

Figure 1 | Both pretrained models are derived from corresponding Gemma pretrained models.

Pretraining

Training Data

All CodeGemma v1.0 models are further trained on 500 billion tokens of primarily English language data from web documents, mathematics, and code. The 2B v1.1 model is trained on 1 trillion tokens. All 2B models are trained with 100% code while the 7B models are trained with a 80% code-20% natural language mixture. Our code corpus comes from publicly available code repositories. Datasets are deduplicated and filtered to remove contamination of evaluation code and certain personal and sensitive data. In addition to the processing done for Gemma, we perform additional pretraining steps for code data.

Preprocessing for Fill-in-the-Middle

The pretrained CodeGemma models are trained using a method based on the fill-in-the-middle (FIM) task (Bavarian et al., 2022) with improvements that address the shortcomings cited in the original work as well as empirically-found systemic issues with existing FIM-trained models. The FIM rate is at 80% in most models, except the pretrained 2B v1.1 where it is at 90%. The relevant formatting control tokens are presented in Table 1. The models are trained to work with both PSM (Prefix-Suffix-Middle) and SPM (Suffix-Prefix-Middle) modes. Figure 2 shows a sample snippet formatted in PSM. We make detailed FIM usage instructions in the Inference Recommendations section.

Context	Relevant Token
FIM prefix	`<|fim_prefix|>`
FIM middle	`<|fim_middle|>`
FIM suffix	`<|fim_suffix|>`
File separator	`<|file_separator|>`

Table 1 | Formatting control tokens used for FIM task. Note that | is the standard pipe character (ASCII code 124).

Multi-file Packing

Many downstream code-related tasks involve generating code based on a repository-level context as opposed to a single file. To improve model alignment with real-world applications, we create training examples by co-locating the most relevant source files within code repositories and best-effort grouping them into the same training examples. Specifically, we employ two heuristics: dependency graph-based packing and unit test-based lexical packing.

To construct the dependency graph, we first group files by repository. For each source file, we extract imports from the top N lines and perform suffix matching to determine the longest matching paths within the repository structure. We determine edge importance (a heuristic measure) between files, and remove unimportant edges to break cyclic dependencies (common in Python). We then calculate all-pairs shortest paths within the graph, where shorter distances signify stronger file relationships. Finally, we linearize the graph of files using a topological sort, selecting the next unparented node based on minimum distance to sorted nodes and using lexicographic order to break ties.

Files not covered by this dependency graph method are sorted alphabetically within their repository with unit tests packed next to their implementations (e.g. `TestFoo.java` beside `Foo.java`).

Instruction Tuning

Our training data consists of a combination of open-source math datasets and synthetically generated code, in addition to the finetuning datasets used by Gemma. By exposing the model to mathematical problems, we aim to enhance its logical reasoning and problem-solving skills, which are essential for code generation.

The instruction-tuned 7B v1.1 model differs from its 1.0 cousin in the reinforcement learning algorithm used (based on Gemma 1.1) and specifics of synthetic data generation. They all follow the general direction below.

Mathematics Datasets

To enhance the mathematical reasoning capabilities of coding models, we employ supervised finetuning on a diverse set of mathematics datasets, including:

MATH Dataset A collection of 12,500 challenging mathematical problems from competitions, providing step-by-step solutions for training models in answer derivation and explanation generation (Hendrycks et al., 2021).

GSM8k Dataset A collection of 8,500 grade school math problems. This dataset tests

```
path/to/the/first/file.py↵
<|fim_prefix|>from typing import List↵
↵
def mean_absolute_deviation(numbers: List[float]) -> float:↵
    """For a given list of input numbers, calculate Mean Absolute Deviation↵
    around the mean of this dataset.↵
    Mean Absolute Deviation is the average absolute difference between each↵
    element and a centerpoint (mean in this case):↵
    MAD = average | x - x_mean |↵
    >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])↵
    1.0↵
    """↵
<|fim_suffix|><|fim_middle|>    return sum(abs(x - mean) for x in numbers) / len(numbers)↵
<|file_separator|>path/to/the/second/file.py↵
<|fim_prefix|>...
```

Figure 2 | Example code snippet in PSM mode. The green ↵ characters are part of the format, whereas uncolored ↵ is from the source. The shown code sample is from HumanEval (Chen et al., 2021).

the multi-step reasoning abilities of models, highlighting their limitations despite the simplicity of the problems (Cobbe et al., 2021a).

MathQA Dataset A large-scale dataset of math word problems (Amini et al., 2019) with annotations built on top of the AQuA dataset (Ling et al., 2017).

Synthetic Mathematical Data A programmatically-generated dataset of algebraic problems used to improve ability to solve long algebra problems.

By leveraging these diverse datasets, we expose the model to a wide range of mathematical problems, increasing their ability to perform complex mathematical reasoning. Our training experiments indicate that these datasets significantly boost code generation performance.

Coding Dataset

Effectively instruction-tuning large language models for code generation tasks requires a substantial amount of question-answer pairs. We leverage synthetic code instruction data generation to create datasets used in the supervised-finetuning (SFT) and reinforcement learning from human feedback (RLHF) phase. We apply the following steps:

Example Generation Following the approach outlined in the OSS-Instruct paper (Wei et al., 2023), we generate a set of self-contained question-answer pairs.

Post-Filtering We filter question-answer pairs using an LLM tasked with evaluating the helpfulness and correctness of the generated question-answer pairs.

Evaluation

We evaluate CodeGemma for code completion and generation performance, as well as natural language understanding, with automated benchmarks across a variety of domains.

Infilling Capability

HumanEval Infilling

The CodeGemma models are trained for code completion purposes. We use the single-line and multi-line metrics in the HumanEval Infilling benchmarks introduced in Fried et al. (2023) to evaluate. Performance against other FIM-aware code models is shown in Table 2.

We observe that our 2B pretrained model is an excellent well-rounded model for code completion use cases, where low latency is a critical factor. It performs on par with the other models while being, in many cases, nearly twice as fast during inference. We attribute this speedup to the base Gemma architectural decisions.

Real-world Evaluation

We validate our model's infilling abilities by masking out random snippets in code with cross-file dependencies, generating samples from the model, and retesting the code files with the generated snippets to show that it performs as expected, a similar approach to Liu et al. (2023) or Ding et al. (2023). Due to our inclusion of very recently committed open source code, we do not use the evaluations directly, but use an internal version with the same testing methodology.

In addition to evaluating on offline evaluations, the model was tested within live coding environments to benchmark its performance against current Google completion models.

Coding Capability

Python Coding

The canonical benchmarks used in coding evaluation are HumanEval (Chen et al., 2021) and Mostly Basic Python Problems (Austin et al., 2021). We present our results in Table 3.

Benchmark	HumanEval	MBPP
2B-PT	31.1%	43.6%
2B-PT 1.1	**37.8%**	**49.2%**
Gemma 2B PT	22.0%	29.2%
7B-PT	44.5%	**56.2%**
7B-IT	56.1%	54.2%
7B-IT 1.1	**60.4%**	55.2%
Gemma 7B PT	32.3%	44.4%

Table 3 | Python coding capability of CodeGemma on de-facto coding benchmarks.

Compared to the base Gemma models (Gemma Team et al., 2024), CodeGemma models perform significantly better on tasks from the coding domain.

Multi-lingual Benchmarks

BabelCode (Orlanski et al., 2023) is used to measure the performance of CodeGemma on code generation across a variety of popular programming languages. Results are presented in Table 4.

Language Capability

We evaluate performance on a variety of domains including question answering (Bisk et al., 2019; Clark et al., 2019, 2018; Joshi et al., 2017), natural language (Hendrycks et al., 2020; Sakaguchi et al., 2019; Zellers et al., 2019) and mathematical reasoning (Cobbe et al., 2021b; Hendrycks et al., 2021). We present the results of our two 7B models next to the instruction-tuned Gemma 7B model in Figure 3.

CodeGemma retains most of the same natural language capabilities seen in the base Gemma models. CodeGemma PT and IT both outperform Mistral 7B (Jiang et al., 2023) by 7.2% and Llama-2 13B model (Touvron et al., 2023) by 19.1% (numbers reported in Gemma Team et al. 2024). Further, we compare scores for GSM8K and MATH in Table 5 from several code models in the 7B size class, and show that CodeGemma excels at mathematical reasoning compared to similarly sized models.

Model	GSM8K	MATH
CodeGemma PT	44.2%	19.9%
CodeGemma IT	41.2%	20.9%
CodeGemma IT 1.1	47.3%	22.3%
Code Llama	13.0%	
DeepSeek Coder	43.2%	19.2%
StarCoder2	40.4%	

Table 5 | Math reasoning capability of other code models in the same 7B size class. Results collected from Guo et al. (2024); Lozhkov et al. (2024); Rozière et al. (2024).

Practical Considerations

CodeGemma is tailored for practical use and deployment in latency-sensitive settings. The 2B

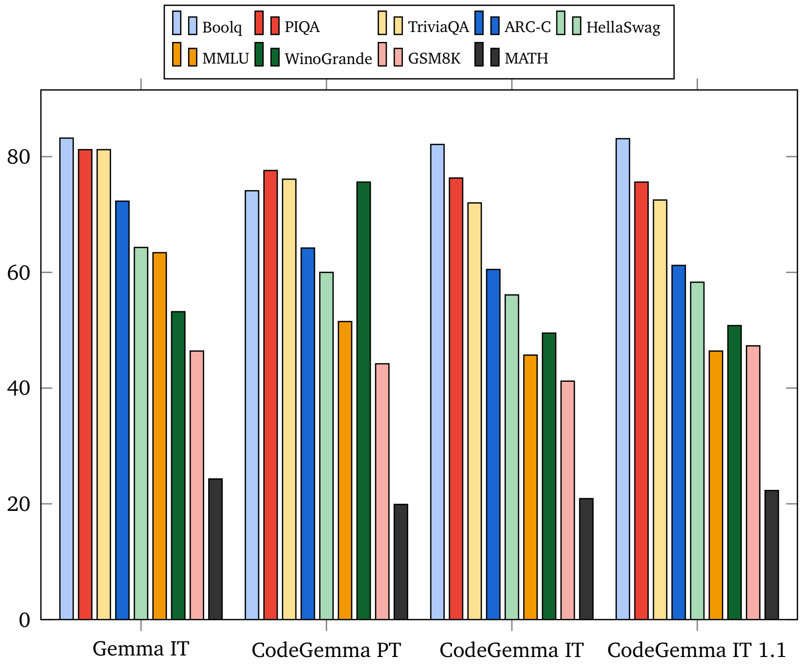

Figure 3 | Language capability comparison of CodeGemma and the instruction-tuned version of Gemma. Both Gemma and CodeGemma are in the 7B size class.

model is considerably faster than all models in our comparison set, which is critical for latency-sensitive applications such as code completion. This speedup does not come with a significant, measured compromise in quality according to our evaluations — the 2B model performs as well or better compared to other open models in its class at code infilling tasks. Consequently, CodeGemma 2B is exceptionally suitable for utilization within Integrated Development Environments (IDEs), local environments, and other applications with memory constraints.

The 7B models, characterized by their strong performance, are general coding models that surpass the baseline Gemma models in terms of coding tasks while maintaining a high level of natural language comprehension. The larger memory requirement during inference renders these models particularly suitable for deployment in hosted environments and applications where model quality is of utmost importance.

The Responsible Deployment section in Gemma Team et al. (2024) contains a thorough discussion about the limitations and benefits of using an open model.

Inference Recommendations

For pretrained models, prompts should be formatted for code completion tasks such as function completion, docstring generation, and import suggestion. Figure 4 shows an example of a prompt format, where the file path is optional but recommended. The stopping strategy for model outputs should be chosen carefully to align with the deployment setting. The most straightforward method is to truncate upon generating a FIM sentinel token, as shown in Table 1.

```
path/file.py↵
<|fim_prefix|>prefix<|fim_suffix|>suffix
<|fim_middle|>
```

Figure 4 | Prompt in PSM mode. The carriage return ↵ is part of the format. There are no spaces after the suffix.

The same formatting as Gemma, with `<start_of_turn>` and `<end_of_turn>` tokens, can also prompt the instruction-tuned model.

Conclusion

We present a collection of open models specialized for coding applications, built on top of Gemma, an openly available family of language models (Gemma Team et al., 2024). These models push the state of the art in code completion and generation, while retaining natural language capabilities from the base models.

The CodeGemma models presented in this report are highly capable language models designed for effective real-world deployment, optimized to be run in latency-constrained settings while delivering high-quality code completion on a variety of tasks and languages. We show that the lessons and technologies that built Gemini and Gemma are transferable to downstream applications, and we are excited to release these models to the broader community and to enable the applications which will be built on top of these models.

		Time (s)		Performance	
	Model	Single	Multi	Single	Multi
2B class	CodeGemma	**543**	**8479**	78.41%	**51.44%**
	CodeGemma 1.1	771	9372	79.28%	50.99%
	DeepSeek Coder	990	13138	79.96%	50.95%
	DeepSeek Coder Instruct	5632	31505	**81.41%**	37.35%
	StarCoder2	3665	20629	77.44%	47.65%
7B class	CodeGemma	**1505**	22896	76.09%	58.44%
	CodeGemma Instruct	8330	49438	68.25%	20.05%
	CodeGemma Instruct 1.1	7579	46888	77.44%	23.66%
	Code Llama*			74.10%	48.20%
	CodeQwen 1.5	7046	41345	32.72%	13.33%
	DeepSeek Coder	1559	**22387**	85.87%	**63.20%**
	DeepSeek Coder Instruct	9500	53498	**86.45%**	58.01%
	StarCoder2	8080	45459	81.03%	53.21%

Table 2 | Single-line and multi-line code completion capability of CodeGemma compared to other FIM-aware code models (Bai et al., 2023; Guo et al., 2024; Lozhkov et al., 2024; Rozière et al., 2024). Time is the total number of seconds to obtain 128-token continuations per each HumanEval Infilling task (1033 tasks in single-line and 5815 multi-line). Measurements are done with HuggingFace's Transformers (Wolf et al., 2020) model implementations on `g2-standard-4` GCE instances with bfloat16 datatype and batch size of 1. * Code Llama numbers are taken from Rozière et al. (2024).

	Language	2B	2B 1.1	7B	7B-IT	7B-IT 1.1
HumanEval	C/C++	**24.2%**	19.9%	32.9%	42.2%	**46.6%**
	C#	10.6%	**26.1%**	22.4%	26.7%	**54.7%**
	Go	**20.5%**	18.0%	21.7%	28.6%	**34.2%**
	Java	29.2%	**29.8%**	41.0%	48.4%	**50.3%**
	JavaScript	21.7%	**28.0%**	39.8%	46.0%	**48.4%**
	Kotlin	28.0%	**32.3%**	39.8%	**51.6%**	47.8%
	Python	21.7%	**36.6%**	42.2%	48.4%	**54.0%**
	Rust	**26.7%**	24.2%	34.1%	36.0%	**37.3%**
MBPP	C/C++	**47.1%**	38.9%	53.8%	56.7%	**63.5%**
	C#	28.7%	**45.3%**	32.5%	41.2%	**62.0%**
	Go	**45.6%**	38.9%	43.3%	46.2%	**53.2%**
	Java	41.8%	**49.7%**	50.3%	57.3%	**62.9%**
	JavaScript	**45.3%**	45.0%	58.2%	**61.4%**	**61.4%**
	Kotlin	46.8%	**49.7%**	54.7%	59.9%	**62.6%**
	Python	38.6%	**52.9%**	59.1%	**62.0%**	60.2%
	Rust	45.3%	**47.4%**	52.9%	**53.5%**	52.3%

Table 4 | Multi-lingual coding capability of CodeGemma (CG) on BabelCode-translated HumanEval and Mostly Basic Python Problems (MBPP) datasets. IT stands for instruction-tuned.

Contributions and Acknowledgments

Core Contributors

赵赫日 (Heri Zhao)
許嘉倫 (Jeffrey Hui)
Joshua Howland
Nguyễn Thành Nam[1] (Nam Nguyen)
左斯琦 (Siqi Zuo)

Contributors

胡琪恩 (Andrea Hu)
Christopher A. Choquette-Choo
Jingyue Shen
Joe Kelley
क्षितिज बंसल (Kshitij Bansal)
Luke Vilnis
Mateo Wirth
Paul Michel
Peter Choy
प्रतिक जोशी (Pratik Joshi)
Ravin Kumar
سرمد ہاشمی (Sarmad Hashmi)
शुभम अग्रवाल (Shubham Agrawal)
Zhitao Gong

Product Management

Євгенія Файн (Jane Fine)
Tris Warkentin

Program Management

Ale Jakse Hartman

Executive Sponsors

Bin Ni
Kathy Korevec
Kelly Schaefer
Scott Huffman

Other Specialty Areas

Special thanks and acknowledgments to these individuals for their assistance in respected areas:

Central Support David Huntsperger, Elisa Bandy, Emma Yousif, Glenn Cameron, James Freedman, Joe Fernandez, Josh Woodward, Keelin McDonell, Minh Giang

Checkpoint Conversions Austin Huang, Matthew Watson, Michael Butler, Michael Moynihan, Morgane Rivière, Phil Culliton, 单志昊 (Zhihao Shan)

Ethics and Safety Antonia Paterson, அநிருத் ஸ்ரீராம் (Anirudh Sriram), Jenny Brennan, 帅杰 (Kevin Shuai), Ludovic Peran, 김민 (Min Kim)

Evaluations अरोमा महेन्द्रू (Aroma Mahendru), ध्वनिक विरड़िया (Dhvanik Viradiya), गौरांग कोठिया (Gaurang Kothiya), हेतुल पटेल (Hetul Patel), Jasmine George, Navneet Potti, संजना पुरोहित (Sanjana Purohit), उत्कर्ष पंड्या (Utkarsh Pandya)

Gemma Model Alek Andreev, Kathleen Kenealy, సూర్య భూపతిరాజు (Surya Bhupatiraju)

Go-to-Market Gabriel Rasskin, Kat Black, Lav Rai, Luiz Gustavo Martin, Manvinder Singh, Meg Risdal, 박민우 (Minwoo Park), Nesh Devanathan, Nilay Chauhan, రాజ్ గుండ్లూరు (Raj Gundluru)

Partnering Ankit Patel, Arthur Zucker, Bhavesh Pawar, Chenjie Luo, Christina Young, 孟东 (Dong Meng), Frederic Bastien, Izzy Putterman, Krzysztof Pawelec, لبنى بن علال (Loubna Ben Allal), Marta Stepniewska-Dziubinska, Omar Sanseviero, Pedro Cuenca, Sharan Chetlur, वैभव श्रीवास्तव (Vaibhav Srivastav), Yu-Hsuan Tseng

Reinforcement Learning Johan Ferret, Léonard Hussenot, Nino Vieillard, Olivier Bachem, Pier Giuseppe Sessa, Robert Dadashi, Sertan Girgin

Team Acknowledgements

Our work is made possible by the dedication and efforts of numerous teams at Google. We would like to acknowledge the support from the following teams: AIDA, DevRel, Gemini Infrastructure, Gemini Safety, Gemma, Google Cloud, Google Research Responsible AI, Kaggle, Keras.

References

A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms, 2019. URL http://arxiv.org/abs/1905.13319.

J. Austin, A. Odena, M. I. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. J. Cai, M. Terry, Q. V. Le, and C. Sutton. Program synthesis with large language models.

[1] Lead.

CoRR, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen. Efficient training of language models to fill in the middle, 2022.

Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi. PIQA: reasoning about physical commonsense in natural language. *CoRR*, abs/1911.11641, 2019. URL http://arxiv.org/abs/1911.11641.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

C. Clark, K. Lee, M. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *CoRR*, abs/1905.10044, 2019. URL http://arxiv.org/abs/1905.10044.

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021a. URL https://arxiv.org/abs/2110.14168v2.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021b. URL https://arxiv.org/abs/2110.14168.

Y. Ding, Z. Wang, W. U. Ahmad, H. Ding, M. Tan, N. Jain, M. K. Ramanathan, R. Nallapati, P. Bhatia, D. Roth, and B. Xiang. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion, 2023.

D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W. tau Yih, L. Zettlemoyer, and M. Lewis. Incoder: A generative model for code infilling and synthesis, 2023.

Gemma Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou, A. Tacchetti, A. Bulanova, A. Paterson, B. Tsai, B. Shahriari, C. L. Lan, C. A. Choquette-Choo, C. Crepy, D. Cer, D. Ippolito, D. Reid, E. Buchatskaya, E. Ni, E. Noland, G. Yan, G. Tucker, G.-C. Muraru, G. Rozhdestvenskiy, H. Michalewski, I. Tenney, I. Grishchenko, J. Austin, J. Keeling, J. Labanowski, J.-B. Lespiau, J. Stanway, J. Brennan, J. Chen, J. Ferret, J. Chiu, J. Mao-Jones, K. Lee, K. Yu, K. Millican, L. L. Sjoesund, L. Lee, L. Dixon, M. Reid, M. Mikuła, M. Wirth, M. Sharman, N. Chinaev, N. Thain, O. Bachem, O. Chang, O. Wahltinez, P. Bailey, P. Michel, P. Yotov, P. G. Sessa, R. Chaabouni, R. Comanescu, R. Jana, R. Anil, R. McIlroy, R. Liu, R. Mullins, S. L. Smith, S. Borgeaud, S. Girgin, S. Douglas, S. Pandya, S. Shakeri, S. De, T. Klimenko, T. Hennigan, V. Feinberg, W. Stokowiec, Y. hui Chen, Z. Ahmed,

Z. Gong, T. Warkentin, L. Peran, M. Giang, C. Farabet, O. Vinyals, J. Dean, K. Kavukcuoglu, D. Hassabis, Z. Ghahramani, D. Eck, J. Barral, F. Pereira, E. Collins, A. Joulin, N. Fiedel, E. Senter, A. Andreev, and K. Kenealy. Gemma: Open models based on gemini research and technology, 2024.

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024.

D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *CoRR*, abs/2009.03300, 2020. URL https://arxiv.org/abs/2009.03300.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *CoRR*, abs/1705.03551, 2017. URL http://arxiv.org/abs/1705.03551.

W. Ling, D. Yogatama, C. Dyer, and P. Blunsom. Program induction by rationale generation : Learning to solve and explain algebraic word problems, 2017. URL https://arxiv.org/abs/1705.04146v3.

T. Liu, C. Xu, and J. McAuley. Repobench: Benchmarking repository-level code auto-completion systems, 2023.

A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W.-D. Li, M. Risdal, J. Li, J. Zhu, T. Y. Zhuo, E. Zheltonozhskii, N. O. O. Dade, W. Yu, L. Krauß, N. Jain, Y. Su, X. He, M. Dey, E. Abati, Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. McAuley, H. Hu, T. Scholak, S. Paquet, J. Robinson, C. J. Anderson, N. Chapados, M. Patwary, N. Tajbakhsh, Y. Jernite, C. M. Ferrandis, L. Zhang, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. Starcoder 2 and the stack v2: The next generation, 2024.

G. Orlanski, K. Xiao, X. Garcia, J. Hui, J. Howland, J. Malmaud, J. Austin, R. Singh, and M. Catasta. Measuring the impact of programming language distribution. *arXiv preprint arXiv:2302.01973*, 2023.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code, 2024.

K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. WINOGRANDE: an adversarial winograd schema challenge at scale. *CoRR*, abs/1907.10641, 2019. URL http://arxiv.org/abs/1907.10641.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan,

M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Y. Wei, Z. Wang, J. Liu, Y. Ding, and L. Zhang. Magicoder: Source code is all you need, 2023. URL http://arxiv.org/abs/2312.02120.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence?, 2019.