IotaCode: A Small Code Model Can Be Reinforced to Beat the Bigger One

Anonymous ACL submission

Abstract

Large language models (LLMs) are one of the most rapidly developing areas of research in machine learning. To fine-tune LLMs to better align with user requests and values, reinforcement learning techniques based on human feedback (RLHF) have been developed, allowing for the inclusion of negative as well as positive examples. An important domain for the application of large language models is the analysis and generation of source code. In this study, we investigated how modern RLHF algorithms can be applied to code generation using the CodeContests problem set. The best results were achieved using the Proximal Policy Optimization algorithm, which significantly improves the supervised fine-tuning baseline, producing IotaCode model with 1.3 billion parameters that surpass the performance of the AlphaCode model with 9 billion parameters.

1 Introduction

In the realm of programming, the path to mastering code development is not solely illuminated by correct solutions; rather, it is also shaped by the trials and errors inherent in the coding process. The educational value of errors, when systematically analyzed and corrected, cannot be overstated. This pedagogical perspective underpins the increasing application of Reinforcement Learning from Human Feedback (RLHF) methodologies, particularly the Proximal Policy Optimization (PPO) algorithm, in training language models for code generation. These models are uniquely positioned to not only produce solutions but also learn from their own inaccuracies, thus mimicking a more authentic learning experience akin to that of a human programmer.

The efficacy of these models, particularly in their ability to learn from both successful outcomes and mistakes, hinges on the robustness of the "teacher" or reward model. This model is trained to distinguish effective and flawed code by evaluating a wide array of coding attempts. Consequently, the development of a capable teacher model requires a substantial corpus of correct and incorrect code samples. In this context, code contest platforms emerge as invaluable resources. These platforms are arenas where programmers continuously contribute solutions to a myriad of problems, generating a rich dataset of both successful solutions and common errors.

By leveraging such datasets, especially erroneous submissions, we can train more nuanced and effective models. These models are adept at not only identifying and rectifying errors, but also guiding the learning process towards a more holistic understanding of coding practices. The overarching goal is to refine the capabilities of language models so that they not only solve problems, but also foster a deeper understanding and proficiency in the programmer using them, thereby enhancing the educational journey of learning through mistakes.

The contributions of this work are two-fold:
(i) we present a relatively small IotaCode[1] 1.3B model for code generation which outperform AlphaCode 9B model on the hard test set of CodeContests and
(ii) we demonstrate that PPO algorithm with reward model can be effectively applied to the code generation domain, unlike in previous works, while novel alignment algorithms show no improvement over supervised fine-tuning.

2 Dataset

Data sources for this work include the CodeContests dataset (Li et al., 2022) and a collection of public solutions from the Codeforces website, along with metadata. CodeContests consists of texts and solutions for 13,328 problems from several

[1]Traditionally the letter ι (reading "iota") is associated with something very small, since it is visually the smallest letter in Greek aplhabet.

sites (Aizu, AtCoder, CodeChef, Codeforces, HackerEarth). There are a total of 4.4 million correct solutions and 8.7 million incorrect solutions. The dump contains solutions for 6,998 problems from 1,534 programming contests. The test set problems were excluded from the dump to ensure proper evaluation.

Supervised fine-tuning requires data to be formatted as prompt and completion pairs. In the context of generating programs, prompts are problem statements and completions are solutions. To promote diversity, we mixed the CodeContests and dump data in approximately equal ratios, resulting in a total of 1.2 million samples. The average sequence length is 1500 tokens. To save GPU memory and eliminate outliers, we only kept samples shorter than 4000 tokens, corresponding to the 99th percentile.

Alignment algorithms require data to be formatted as triplets consisting of a prompt, a preferred completion, and a dispreferred completion. Therefore, the CodeContests dataset is a natural choice due to the presence of incorrect solutions. Preferred completions are those with an "OK" verdict, while all other completions are considered dispreferred. We tested two strategies for collecting triplets: the first was to select solutions closest in terms of Levenshtein distance, and the second was to simply assign a random wrong solution to each correct one. We did not observe significant differences between these two approaches. Therefore, we mixed random and closest negatives, resulting in a total of 2 million triplets.

3 Method

Traditionally, the alignment of language models (Christiano et al., 2017) is based on the training objective, specified as:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \Big[r_\phi(x,y)\Big] - \beta\mathbb{D}_{\mathrm{KL}}\Big[\pi_\theta(y|x)||\pi_{\mathrm{ref}}(y|x)\Big], \quad (1)$$

where $\mathcal{D}$ is the training dataset, x is the prompt, y is the completion, π_θ is the policy (language model) being optimized, and $r_\phi(x,y)$ is the scalar output of the reward model for prompt x and completion y, π_{ref} is the reference model (usually obtained by supervised fine-tuning pre-trained LLM).

Conventionally (Ouyang et al., 2022), the optimization of the objective 1 is handled by reinforcement learning, especially by the Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm. PPO is an on-policy actor-critic reinforcement learning algorithm, designed to prevent large, destabilizing changes, ensuring smoother training by using a clipped surrogate objective. At first a reward model (RM) is trained to match human preferences on a dataset of ranked response pairs by optimizing the log likelihood of a Bradley-Terry model (Bradley and Terry, 1952)

$$\min_\phi -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma(r_\phi(x,y_w)-r_\phi(x,y_l))\right], \quad (2)$$

where y_w and y_l are preferred and unpreffered responses respectively. Afterwards, the LLM is trained to generate responses maximizing the RM's values.

Despite these successes, reinforcement learning algorithms face inherent limitations such as inefficiency, instability, extensive resource requirements, and complex hyperparameter tuning, which can impede the performance and scalability of LLMs. To overcome these challenges, recent studies have introduced various variants of RL-free methods that do not rely on PPO.

A recent development by Direct Preference Optimization (Rafailov et al., 2023) introduced an RL-free approach aimed at aligning a policy model by optimizing the likelihood of the preferred and unpreferred responses. The DPO loss function is mathematically articulated in Equation 3 as follows:

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta;\pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\mathrm{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\mathrm{ref}}(y_l|x)}\right)\right]. \quad (3)$$

Despite DPO being more stable and easier to implement than PPO, it still requires separate supervised fine-tuning phase and calling reference model during training. Addressing these limitations an even simpler approach named Odds Ratio Preference Optimization (ORPO) (Hong et al., 2024) was introduced, which efficiently penalizes the model from learning undesired generation styles during SFT by adding odd ratio penalty term to the language modelling loss.

Loss function is formulated as follows:

$$\mathcal{L}_{\mathrm{ORPO}} = \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}[\mathcal{L}_{\mathrm{SFT}} + \lambda\mathcal{L}_{\mathrm{OR}}], \quad (4)$$

where

$$\mathcal{L}_{\mathrm{OR}} = -\log\sigma\left(\log\frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)}\right), \quad (5)$$

where $\mathcal{L}_{\text{SFT}}$ is standard supervised loss (Eq. 7) and

$$\textbf{odds}_\theta(y|x) = \frac{p_\theta(y|x)}{1 - p_\theta(y|x)}. \quad (6)$$

4 Experiments

For our experiments, we use DeepSeek Coder 1.3B (Guo et al., 2024) as the base model. This model is a transformer decoder-only architecture, pretrained on a large corpus of code and natural language.

SFT First phase is supervised fine-tuning (SFT), where the model is trained to generate correct solution by optimizing standard language modelling loss:

$$\mathcal{L}_{SFT} = \sum_i \log p_\theta(y_i|x, y_{<i}), \quad (7)$$

where y_i refers to i-th solution token, x is prompt (problem text), $y_{<i}$ are tokens before y_i.

Due to the high sequence length (4000 max), the batch size was set to 1, and gradients were accumulated over 4 steps. Considering parallelism and gradient accumulation, the effective batch size was 8. AdamW (Loshchilov and Hutter, 2018) was used for optimization with a learning rate of 1e-4, a cosine annealing decay schedule (Loshchilov and Hutter, 2016), and 5000 linear warm-up steps.

DPO Fine-tuning all parameters with DPO significantly degraded the model's performance. To introduce additional regularization into the training process, we used LoRA (Hu et al., 2021) with a rank of 32, applied to all layers except the embedding layer. It is worth noting that using LoRA during DPO significantly reduces memory consumption, as there is no need to keep a separate reference model in memory, one can simply disable adapter parameters during forward pass. For greater stability, in accordance with the original paper, the effective batch size was increased to 64, the learning rate was set to 1e-6, and β was set to 1.0. The remaining parameters were the same as in SFT training.

ORPO ORPO training hyperparameters are similar to DPO, excepting $\beta = 0.1$.

Reward Model Training The reward model plays a crucial role by assessing the relative quality of generated code, which is central to our training approach. Training this model presents unique challenges, especially in the context of programming contests:

- **Challenge of Plateauing:** During the training phase, we observed a plateau in the loss and performance metrics, as the model struggled to differentiate effectively between higher and lower quality solutions.

- **Domain Adaptation Strategy:** To address this, a domain adaptation approach was adopted by pretraining the reward model in a language modeling setup using code contest data. This strategy helped prevent plateauing by acquainting the model with the typical structures and logic found in contest problems.

- **Training Details:** The model was trained with a learning rate of 1.4×10^{-5} over a single epoch, using a dataset comprising 2,000,000 samples, ensuring robust assessment capabilities.

PPO Proximal Policy Optimization is utilized to iteratively refine the policy model based on the feedback from the reward model. This method is designed to enhance the policy's ability to generate higher-quality code by aligning it more closely with the reward evaluations. Training with PPO involves:

- **LoRA Adapters:** Low-Rank Adapters (LoRA) are utilized to restrict the model's flexibility, thus maintaining control over the learning process and preventing overfitting. LoRA is applied with a rank of 32 to all layers except the embedding layer.

- **Training Details:** The effective batch size is set to 28, with a learning rate of 1.4×10^{-5}. A cosine annealing decay schedule is used, with training conducted over 10 epochs given the dataset's size of about 8,000 unique problems.

Evaluation The problems in CodeForces competitions are significantly more challenging than those in HumanEval (Chen et al., 2021) or MBPP (Austin et al., 2021), requiring the recognition and use of non-trivial algorithms. Therefore, following the approach of AlphaCode (Li et al., 2022), we generate 1,000 solutions for each problem. The test set of the CodeContests dataset consists of 165 problems, each with 2-3 public tests and 200 private tests. For efficient generation of a large number of solutions, we use the VLLM server[2] with PagedAttention (Kwon et al., 2023) support and automatic

[2] https://github.com/vllm-project/vllm

	# solved problems	compile rate
AlphaCode 9B	24	
AlphaCode 1B	20	
DeepSeek 1.3B-instruct	15	
SFT	21	78.7
SFT + DPO	21	78.8
SFT + ORPO	21	77.3
SFT + PPO	**25**	40.3

Table 1: Performance on the CodeContests test split having 165 problems.

batching of asynchronous requests. This setup allows us to generate 165,000 solutions with a 1.3B model in approximately 30 hours.

The results are presented in Table 1. Reward-free algorithms DPO and ORPO did not improve the model's performance. The best results were achieved by the model trained with PPO, which successfully solved 4 more problems than the SFT model and surpassed the performance of the AlphaCode 9B model, despite having 7 times fewer parameters. We believe that the superiority of PPO in generating solutions for algorithmic problems is due to PPO being an online algorithm, unlike DPO and ORPO. We leave the theoretical explanation of this observation for future work.

5 Related Work

Initial experiments with large language models like GPT-Neo (Black et al., 2022) and GPT-J (Wang and Komatsuzaki, 2021) revealed that incorporating code into the training data enables program synthesis, even with medium-sized models. Concurrently, specialized models aimed at code comprehension and program synthesis from natural language prompts have been developed. These include CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), Codex (Chen et al., 2021), CodeT5 (Wang et al., 2021), UnixCoder (Guo et al., 2022), CodeGen (Nijkamp et al., 2023), StarCoder (Lozhkov et al., 2024), and phi-1 (Gunasekar et al., 2023). AlphaCode (Li et al., 2022) demonstrated the ability of language models to efficiently solve even competitive level coding problems.

Source code can be evaluated in terms of functional correctness and compilability. CodeRL (Le et al., 2022) and PPOCoder (Shojaee et al., 2023) use actor-critic deep reinforcement learning to directly optimize these aspects by running generated code against test cases.

6 Conclusion

This study presented a comprehensive examination of the application of reinforcement learning with human feedback (RLHF) algorithms to the field of code generation, focusing on the challenges of programming contests. We specifically explored how the Proximal Policy Optimization (PPO) algorithm can be utilized to enhance the capabilities of large language models in generating code that aligns more closely with human expert performance. The experimental results demonstrate that integrating feedback mechanisms through the reward model substantially improves the model's ability to generate viable code solutions, ultimately surpassing traditional supervised learning methods. In contrast, novel reward-free algorithms DPO and ORPO show no improvement over SFT.

The findings suggest that the careful integration of reinforcement learning techniques can lead to significant advancements in the performance of models designed for code generation. Moreover, the study highlights the potential of combining various training paradigms to overcome inherent limitations in each method, thereby setting a new standard for model training in applied AI fields.

7 Limitations

Despite the successes reported, this study acknowledges several limitations that warrant consideration:

- **Resource Intensity:** The training methodologies employed, particularly PPO and the use of reward models, are resource-intensive, requiring substantial computational power and data, which might not be feasible in all research or practical contexts.

- **Generalizability:** While the results are promising within the scope of programming contests, the generalizability of these models to other domains of code generation or different programming languages has not been thoroughly explored.

- **Dependency on Quality Data:** The performance of the models heavily relies on the availability and quality of the training data. In environments where high-quality or domain-specific data is scarce, the models' effectiveness could be significantly diminished.

- **Complexity in Implementation:** The implementation of combined training approaches, such as integrating LoRA with PPO, introduces complexity that can increase the difficulty of model tuning and maintenance.

- **Potential for Overfitting:** There is a risk of overfitting when fine-tuning with highly specific data sets such as those from code contests, potentially limiting the model's performance on novel or unseen problems.

Future work will aim to address these limitations by exploring more efficient training algorithms, expanding the applicability of models to a broader range of programming tasks, and enhancing the models' ability to generalize from limited or noisy data. Additionally, efforts will be made to streamline model architectures to reduce resource consumption without compromising the quality of the generated code.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *arXiv preprint*. ArXiv:2108.07732 [cs].

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model. *Preprint*, arXiv:2204.06745.

Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Inclomplete Block Design: The Method of Paired Comparisons. *Biometrika*, 39(3-4):324–345.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint*. ArXiv:2107.03374 [cs].

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks are all you need. *Preprint*, arXiv:2306.11644.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. *arXiv preprint*. ArXiv:2203.03850 [cs].

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. *arXiv preprint*. ArXiv:2009.08366 [cs].

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. *arXiv preprint*. ArXiv:2401.14196 [cs].

Jiwoo Hong, Noah Lee, and James Thorne. 2024. ORPO: Monolithic Preference Optimization without Reference Model. *arXiv preprint*. ArXiv:2403.07691 [cs].

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 21314–21328. Curran Associates, Inc.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with restarts. *ArXiv*, abs/1608.03983.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. *arXiv preprint*. ArXiv:2402.19173 [cs].

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. *Preprint*, arXiv:2203.13474.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. 2023. Execution-based code generation using deep reinforcement learning. *Preprint*, arXiv:2301.13816.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. *arXiv preprint*. ArXiv:2109.00859 [cs].