Efficient Post-training of LLMs for Code Generation With Offline Reinforcement Learning

Anonymous Authors[1]

Abstract

Post-training using online reinforcement learning (RL) is an important training step for LLMs, including code-generating models. However, online RL for code generation involves LLM inference and verification of the generated output, which can take considerable time and resources. In this paper, we explore the application of offline RL to code-generating models by leveraging existing code datasets. Our experiments demonstrate that offline RL is an effective training strategy for improving LLM performance. We show that offline RL can be especially beneficial for small LLMs and challenging coding problems.

1. Introduction

Training LLMs with feedback based on specific human preferences is a key aspect of Large Language Model (LLM) training to improve LLM performance and make its output more desirable to humans. In many areas, such as software engineering and math, these preferences are based on verified rewards. In software engineering, the commonly used method for verification is functional correctness of generated code.

Numerous Reinforcement Learning (RL) algorithms, such as PPO (Schulman et al., 2017), GRPO (Shao et al., 2024), RLOO (Kool et al., 2019; Ahmadian et al., 2024), and their variants have been proposed to train LLMs with preferences. However, they are on-policy online learning algorithms; training LLMs with these algorithms is bottle-necked due to the inefficient transformer inference. To address inefficient inference, inference engines such as SGLang or vLLM (Kwon et al., 2023) are typically employed. However, using these engines makes RL training off-policy (Yao et al.) because the model's logits differ between training and inference. The gain in performance under off-policy setting shows that on-policy algorithms can be directly applied in an off-policy setting, at least for LLMs.

In this work, we go one step further and study the impact of online policy gradient algorithms in an offline setup. That is, we train models on pre-existing data rather than sampling new data from the model. This training setup addresses three major limitations of online learning of LLMs.

Efficiency. Offline training eliminates the need to sample new data and does not require the verification of code samples, which can significantly speed up the RL training.

Diversity. Offline RL can address another major issue in online LLM training – entropy collapse (Yu et al., 2025). As online training of LLMs progresses, the diversity of generated outputs decreases, leading to reduced exploration and, hence, reduced gains from further training. Using pre-existing data in offline training ensures sufficient diversity within a group. We can also use heuristics to control the diversity in a batch.

Training Stability. During online RL training of LLM, if all the generated solutions to a particular problem are correct, the variance in the training increases, resulting in no improvement or degradation in the performance of the LLM. Using offline data can also help alleviate this issue.

Thus, training LLMs with offline data can help improve the post-training phase. These gains motivate our exploration of offline RL for LLMs in the context of code generation.

Our study reveals that,

- Online policy gradient algorithms when applied directly in an offline setup can improve LLMs in code generation.
- Offline RL can improve the model on challenging programming problems.
- Training with offline RL also improves pass@10, especially on harder problems showing that offline RL does not lead to reduced diversity in generated code.

Our study also establishes some failure cases of using offline RL. When the base model is already good at a task, offline RL does not improve the model. The gain is also limited

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

in the case of easy problems whereby SFT outperforms offline RL. Further theoretical and empirical analysis to develop new training algorithms or using Q-learning can help alleviate these problems.

2. Training Setup

2.1. Models

For our experiments, we have used Qwen2.5-Coder (Hui et al., 2024). Specifically, we train models at two scales: 0.5B parameters and 7B parameters. For the 0.5B parameter model, we train the full model. The 7B parameter model is trained with LoRA (Hu et al., 2021).

2.2. Dataset

We use the CodeNet dataset (Puri et al., 2021) for training. The CodeNet dataset is a large-scale code dataset comprising 4k problems and 14 million solutions across more than 50 programming languages. More importantly, the CodeNet dataset contains metadata such as functional correctness, syntactic correctness, time taken to run the code, memory consumed, and code size. This metadata can be used to provide the model with feedback on the code's quality. In this work, we have used functional and syntactic correctness. Additionally, we have only trained on Python solutions.

A key limitation of the CodeNet dataset is its high imbalance. The imbalance exists in two dimensions: 1) the number of solutions to a given problem (ranging from 0 to 27k) and 2) the very high proportion of correct solutions. To reduce the imbalance, we limit the maximum number of correct solutions to a given problem to 50.

2.3. Offline RL with Leave one out

Following Ahmadian et al. (2024), we use REINFORCE Leave-one-out (RLOO) (Kool et al., 2019). The RLOO objective is,

$$\nabla_\theta J_{\text{RLOO}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \hat{A}_i \, \nabla_\theta \log \pi_\theta(y_i \mid x) \tag{1}$$

where, π_θ is the policy being trained and $\hat{A}_i$ is the advantage. Ahmadian et al. (2024) samples the sequences $y_1, \ldots, y_n$ from π_θ. In our setup, we can assume that these sequences have been sampled from an unknown policy π_β. The advantage is calculated by using n samples for each problem statement to reduce variance. The advantage in Ahmadian et al. (2024) is calculated as,

$$\hat{A}_i = R(y_{(i)}, x) - \frac{1}{n-1}\sum_{j \neq i} R(y_{(j)}, x) \tag{2}$$

We also experiment with the advantage commonly used with GRPO. It is given by,

$$\hat{A}_i^{GRPO} = \frac{R(y_{(i)}, x) - mean(\mathbf{R})}{std(\mathbf{R})} \tag{3}$$

Nair et al. (2021) proposed an online RL training algorithm that uses offline data augmented with online interactions. While we do not engage in online interaction, we draw inspiration from their proposed algorithm. However, instead of using a separate critic for advantage estimation, we use the GRPO advantage.

The advantage in this case can be represented as,

$$\hat{A}_i^{exp} = exp(\hat{A}_i^{GRPO}) \tag{4}$$

Practically, this results in increasing the weight of correct samples and reducing the weight of negative samples. Thus, it pushes the overall objective closer to SFT while also using negative samples.

2.4. Batches and Groups

We experimented with group sizes of 4 and 8 and found that the models performed better with 4. Thus, we conducted all the experiments with a group size of 4. For each group, we ensured there was at least one correct and one incorrect solution. This condition results in the removal of problems with no correct solutions (usually the most difficult ones). In this sense, it is a very limiting condition, but it reduces the variance in the advantage calculation. In the future, the condition can be relaxed by using variance reduction techniques such as advantage clipping.

2.5. Reward

We provide the following reward based on the status of the code.

$$r = \begin{cases} +1.0 & \text{All Test Cases Passed} \\ -0.1 & \text{Test Cases Failed} \\ -0.5 & \text{Time Limit Exceeded} \\ -0.6 & \text{Runtime Error} \\ -1.0 & \text{Compile Error} \end{cases} \tag{5}$$

2.6. Computational Constraints

While numerous variants of RL training algorithms have been proposed for LLMs in recent years, they require significant computational budget. We apply specific computational constraints to ensure a computationally efficient training. These constraints are: 1) Use of a single A100 80GB GPU for each training run, and 2) A training time for

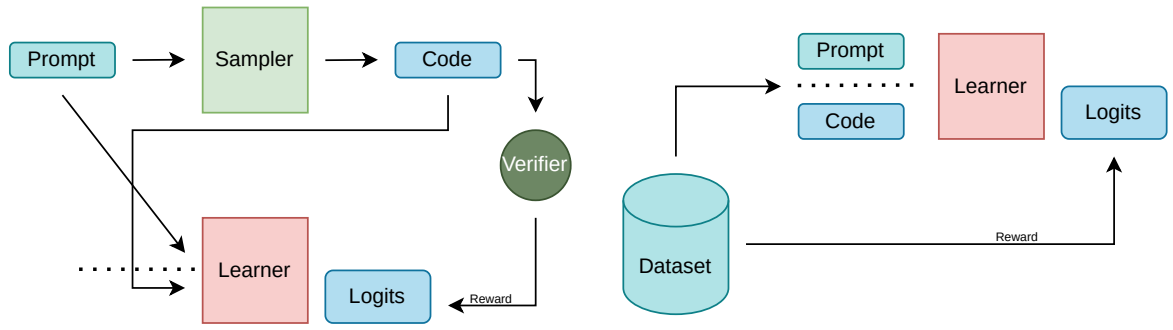

Figure 1. Comparison of typically used setup for training LLM with rewards (left) v/s the proposed offline setup (right). In left, the sampler and learner are the same models but running in different modes for inference and training respectively resulting in deviation between sampling policy and the policy being trained. The proposed setup removes the need for sampling and verification during training, making training faster and efficient.

less than 30 hours for the 7B model and less than 5 hours for the 0.5B model.

To satisfy these constraints, we train the models with a very small batch size of 8 and for a small number of steps. We limit the code length to 2048 tokens. Code samples longer than 2048 tokens are ignored.

2.7. Benchmarks

We evaluate the trained models on two benchmarks: 1) MBPP (Austin et al., 2021) and 2) APPS+ (Dou et al., 2024). MBPP consists of simple Python problems. APPS+ contains problems of different difficulty levels categorized into "introductory", "interview", and "competitive". For MBPP, we report pass@1; for APPS+, we report pass@1 and pass@10 across all three categories.

3. Results and Discussion

We present the results in Table 1, Table 2 and Table 3. We present a discussion around the results in this section.

3.1. Offline RL improves code generation capability.

Across both benchmarks and different difficulty levels of APPS+, we observe that the code generation capability of the LLM improves with offline RL. The gain from offline RL is most consistent when RLOO combines with the GRPO advantage. This gain shows the importance of reducing the variance in advantage estimation. However, we do not observe any improvement in the performance of the 7B model on MBPP. The lack of improvement shows that, on tasks where the base model is already good, offline RL might be ineffective. However, improving the offline data and integrating some online training might help.

3.2. Offline RL improves model on difficult problems.

An important gain from offline training is improvement on difficult interview- and competition-level problems with the larger 7B model and on interview-level problems with the 0.5B model. The improvement is even more stark in the case of competition-level problems for the 7B model. In this case, SFT makes the model worse on both pass@1 and pass@10 while RLOO with GRPO advantage improves the model by 88% on pass@1 and 40% on pass@10.

3.3. Small model requires different scale of reward.

As shown in Table 1, RLOO with exponential advantage leads to significant improvements in the model's performance on introductory and interview-level problems. However, unlike the 7B model, neither RLOO nor RLOO with GRPO advantage improves the model. Using the exponential advantage changes the reward range from $[-1, 1]$ to $[0.37, 2.72]$. Thus, small models might benefit from a different reward level. However, the same reward harms the model on MBPP. The impact of reward across various models and benchmarks also underscores the importance of empirically determining the optimal reward during training.

4. Related Works

Code generation is a very common application for which LLMs are leveraged, and many open source LLMs have been released (Hui et al., 2024; Guo et al., 2024; Jiang et al., 2026). Within this domain, Reinforcement Learning with Verifiable Reward (RLVR) (Lambert et al., 2025) has emerged as an important paradigm, in which feedback is

Table 1. Performance of Qwen-2.5-Coder 0.5B model under different training on the APPS+ dataset. The best performance in a category is **bold**. The performance on competitive problems is trivial across all training setup.

Model	Pass@1			Pass@10		
	Introductory	Interview	Competition	Introductory	Interview	Competition
Base	0.03	0.06	0.00	0.30	0.60	0.00
SFT	**1.94**	1.87	**0.16**	**5.9**	8.1	**0.87**
RLOO	0.06	0.12	0.05	0.5	1.1	0.35
with GRPO advantage	0.07	0.59	0.05	0.4	3.5	0.35
advatnage with exp()	1.67	**2.15**	0.10	5.4	**8.6**	0.70

Table 2. Performance of Qwen-2.5-Coder 7B model under different training on the APPS+ dataset. The best performance in a category is **bold**. If the performace degrades compared to base model, then it is marked in red.

Model	Pass@1			Pass@10		
	Introductory	Interview	Competition	Introductory	Interview	Competition
Base	4.5	7.01	1.42	11.90	25.70	8.22
SFT	**9.21**	11.30	1.38	**16.50**	**30.30**	6.29
RLOO	6.98	10.31	2.06	13.80	27.80	8.04
with GRPO advantage	7.72	**12.14**	**2.67**	14.70	30.20	**11.54**
advatnage with exp()	7.14	11.09	2.38	15.10	**30.30**	9.97

Table 3. Performance of 0.5B and 7B model on MBPP benchmark. The best performance is marked in **bold**. If the performance degrades over the base model, then it is marked in red.

Training	0.5B	7B
Base	36.2	64.4
SFT	35	63.2
RLOO	37.4	64.4
with GRPO Advantage	**39.4**	64
Advantage with exp()	35	64.40

based on verifiable rewards rather than a reward model. However, training is typically done using online learning.

CodeRL (Le et al., 2022) was trained with offline data and verifiable reward. However, the code was generated from the model at the beginning of the training. In this case, there is a risk of lack of good samples if the base model is bad. Additionally, CodeRL was trained with a frozen critic model. Instead, we use human-written code, which ensures good and bad samples and code diversity. There is also no need for a critic model in our setup.

5. Limitations and Future Work

In this paper, we have shown that offline RL is a promising method to train LLMs with verifiable reward for functionally correct code generation. However, our work has some limitations. These limitations are,

- We have not experimented with different learning rate and different rewards. Additionally, we have used a small batch size.
- The dataset is highly imbalanced. While we have used some heuristics to reduce the imbalance, the rules for filtering the data could be improved.
- We have only used offline data. The offline training can be coupled with online interaction to improve exploration.
- We have used on-policy online algorithms directly in offline setup. Developing theoretically-grounded algorithms for offline training of LLMs or using Q-learning (Snell et al., 2023), which might be more suitable in offline case, can help improve the performance further.

Thus, there are different axis along which the training can be improved in future work.

6. Conclusion

In this paper we have shown that offline RL can be an efficient method for post training of code generating LLMs. Offline RL improves both pass@1 and pass@10. The improvement is even more significant on difficult problems. Moreover, the gain in performance has been achieved under severe computational constraints and with a highly imbalanced data. Better reward design and dataset composition can further improve the model. Integrating additional signals, such as efficiency, memory consumption, and security alongside functional correctness can further enhance the quality of generated code. Integrating these characteristics is simpler in the case of offline data compared to evaluating the code for each characteristic in an online setup.

References

Ahmadian, A., Cremer, C., Gallé, M., Fadaee, M., Kreutzer, J., Pietquin, O., Üstün, A., and Hooker, S. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024. URL https://arxiv.org/abs/2402.14740.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Dou, S., Liu, Y., Jia, H., Xiong, L., Zhou, E., Shen, W., Shan, J., Huang, C., Wang, X., Fan, X., Xi, Z., Zhou, Y., Ji, T., Zheng, R., Zhang, Q., Huang, X., and Gui, T. Stepcoder: Improve code generation with reinforcement learning from compiler feedback, 2024. URL https://arxiv.org/abs/2402.01391.

Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y. K., Luo, F., Xiong, Y., and Liang, W. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024. URL https://arxiv.org/abs/2401.14196.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., Dang, K., Fan, Y., Zhang, Y., Yang, A., Men, R., Huang, F., Zheng, B., Miao, Y., Quan, S., Feng, Y., Ren, X., Ren, X., Zhou, J., and Lin, J. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, 35(2):1–72, January 2026. ISSN 1557-7392. doi: 10.1145/3747588. URL http://dx.doi.org/10.1145/3747588.

Kool, W., van Hoof, H., and Welling, M. Buy 4 REINFORCE samples, get a baseline for free!, 2019. URL https://openreview.net/forum?id=r1lgTGL5DE.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., Gu, Y., Malik, S., Graf, V., Hwang, J. D., Yang, J., Bras, R. L., Tafjord, O., Wilhelm, C., Soldaini, L., Smith, N. A., Wang, Y., Dasigi, P., and Hajishirzi, H. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., and Hoi, S. C. H. Coderl: Mastering code generation through pretrained models and deep reinforcement learning, 2022. URL https://arxiv.org/abs/2207.01780.

Nair, A., Gupta, A., Dalal, M., and Levine, S. Awac: Accelerating online reinforcement learning with offline datasets, 2021. URL https://arxiv.org/abs/2006.09359.

Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M., Decker, L., Thost, V., Buratti, L., Pujar, S., Ramji, S., Finkler, U., Malaika, S., and Reiss, F. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks, 2021. URL https://arxiv.org/abs/2105.12655.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Snell, C., Kostrikov, I., Su, Y., Yang, M., and Levine, S. Offline rl for natural language generation with implicit language q learning, 2023. URL https://arxiv.org/abs/2206.11871.

Yao, F., Liu, L., Zhang, D., Donge, C., Shang, J., and Gao, J. Your efficient rl framework secretly brings you off-policy rl training. https://fengyao.notion.site/off-policy-rl. Accessed: 2026-05-07.

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Dai, W., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Yan, L., Qiao, M., Wu, Y., and Wang, M. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.