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

Abhinav Anand [*1] **Mingze Wu** [*1] **Shweta Verma** [1] **Mira Mezini** [1 2 3]

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 large language models (LLMs) with feedback that reflects human preferences is a central component of improving model performance and aligning outputs with user expectations. In domains such as software engineering and mathematics, these preferences are often grounded in verifiable rewards. For example, in software engineering, functional correctness of generated code is the most widely used criterion for validating model outputs.

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 from preference feedback. These methods are inherently on-policy and rely on online data collection. In practice, however, training is often bottlenecked by the high cost of transformer inference. To mitigate this, high-throughput inference engines such as SGLang or vLLM (Kwon et al., 2023) are commonly used. This introduces a discrepancy between training and inference logits, effectively making the training process off-policy (Yao et al.). Empirically, the observed performance gains in this setting suggest that on-policy algorithms can still be successfully applied in mildly off-policy regimes, at least for LLMs.

In this work, we take this observation further by systematically studying the behavior of online policy gradient methods in a fully offline setting. Specifically, we train models on pre-collected datasets instead of sampling new data from the model. This setup addresses several key limitations of online RL for LLMs:

Efficiency. Offline training removes the need for repeated sampling and external verification (e.g., by executing generated code), substantially reducing training cost and latency.

Diversity. Online RL for LLMs often suffers from entropy collapse (Yu et al., 2025), where output diversity decreases over time, limiting exploration and diminishing returns. Offline training mitigates this issue by leveraging pre-existing datasets with inherent diversity; additionally, diversity can be explicitly controlled through data selection or batching strategies.

Training stability. In online settings, when all sampled solutions for a prompt are correct, the resulting low reward variance can lead to unstable or uninformative gradient updates, sometimes degrading performance. Offline data, which naturally includes a mix of solution qualities, helps maintain informative training signals.

Our empirical study shows that:

- Online policy gradient algorithms can be directly applied in an offline setting to improve code generation performance.
- Offline RL yields the largest gains on challenging programming problems.
- Offline training improves pass@10—particularly on harder tasks, indicating that it preserves or even en-

[*]Equal contribution [1]Technische Universitat, Darmstadt, Germany [2]Hessian Center for Artificial Intelligence, Darmstadt, Germany [3]National Research Center for Applied Cybersecurity ATHENE. Correspondence to: Abhinav Anand <abhinav.anand@tu-darmstadt.de>.

Accepted at ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning. Seoul, South Korea. Copyright 2026 by the author(s).

hances output diversity.

At the same time, we identify important limitations. When the base model already performs well, offline RL provides little to no improvement. Similarly, for easier problems, supervised fine-tuning (SFT) can outperform offline RL. These findings suggest the need for improved algorithms, potentially combining offline RL with value-based methods such as Q-learning, or more adaptive training strategies.

2. Training Setup

2.1. Models, Training Method, and Dataset

We use Qwen2.5-Coder (Hui et al., 2024) as our base model. We consider two scales, with 0.5B and 7B parameters. For the 0.5B model, we fine-tune all parameters, whereas for the 7B model we apply LoRA (Hu et al., 2021) adapters instead of full-model training.

We train on the CodeNet dataset (Puri et al., 2021). CodeNet is a large-scale corpus comprising around 4k programming problems and 14 million solutions in more than 50 programming languages, accompanied by rich metadata. This metadata includes functional correctness, syntactic correctness, runtime, memory usage, and code size, which can be used to define reward signals for code quality. In this work, we restrict ourselves to Python solutions and use only functional and syntactic correctness as feedback.

A key limitation of CodeNet is its imbalance. This appears along two axes: (1) the number of solutions per problem, which ranges from 0 to 27k, and (2) a very high proportion of correct solutions. To partially mitigate this, we cap the number of correct solutions per problem at 50.

2.2. Offline RL with Leave one out

Following Ahmadian et al. (2024), we adopt the REINFORCE Leave-one-out (RLOO) objective (Kool et al., 2019), defined as:

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

where, π_θ is the policy being trained and $\hat{A}_i$ denotes the advantage for sample i. In Ahmadian et al. (2024), the sequences $y_1, \ldots, y_n$ are drawn from the current policy π_θ. In our offline setting, 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 RLOO advantage used in Ahmadian et al. (2024) is

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

that is, the reward of sample minus the mean reward of the remaining samples for the same input.

We also experiment with the advantage formulation commonly used in GRPO, given by

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

where R denotes the vector of rewards for all samples of a given problem.

Nair et al. (2021) propose an online RL algorithm that combines offline data augmented with online interactions. Although we do not perform online interaction, our method is inspired by their weighting scheme. However, instead of learning a separate critic for advantage estimation, we adopt the GRPO-style advantage:

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

This weighting increases the relative contribution of high-reward (correct) samples and down-weights low-reward (incorrect) ones. In effect, the objective moves closer to supervised fine-tuning (SFT) while still leveraging negative examples through their reduced, but non-zero, weights.

2.3. Batches and Groups

We experiment with group sizes of 4 and 8 and observe that the models performed better with group size of 4; all subsequent experiments therefore use a group size of 4. For each group, we enforce the presence of at least one correct and one incorrect solution. This constraint excludes problems with no correct solutions – typically the hardest instances – making it somewhat restrictive, but it substantially reduces the variance of the advantage estimates. In future work, this requirement could be relaxed by incorporating additional variance-reduction techniques, such as advantage clipping.

2.4. Reward

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

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

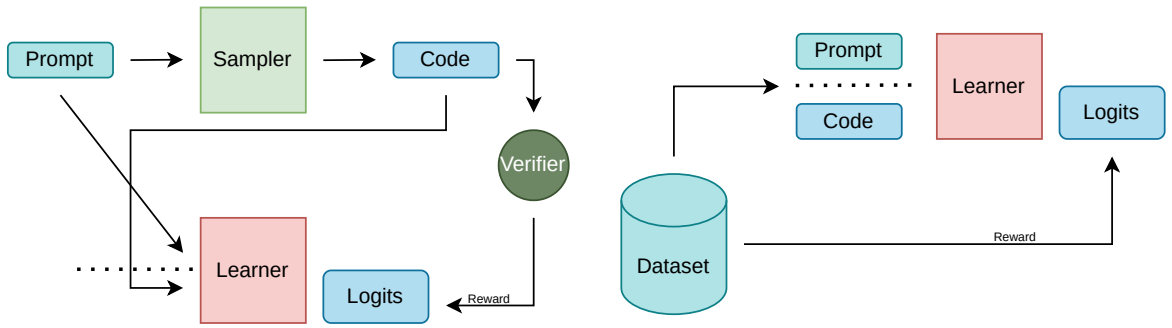

Figure 1. Comparison of typically used setup for training LLM with rewards (left) v/s the proposed offline setup (right). In left, the sampler and learner are the same models but running in different modes for inference and training respectively resulting in deviation between sampling policy and the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

2.5. Computational Constraints

We frame our study under explicit computational constraints. Recent RL algorithms for LLMs typically assume substantial compute budgets, whereas we restrict each training run to a single A100 80GB GPU and limit wall-clock time to under 30 hours for the 7B model and under 5 hours for the 0.5B model.

To meet these constraints, we use a small batch size of 8 and train for a relatively modest number of optimization steps. We further cap the generated code length at 2048 tokens, discarding samples that exceed this limit.

2.6. Benchmarks

We evaluate our models on two benchmarks: 1) MBPP (Austin et al., 2021) and 2) APPS+ (Dou et al., 2024). MBPP consists of simple Python problems, whereas APPS+ includes problems spanning three different difficulty levels: "introductory", "interview", and "competitive". For MBPP, we report pass@1; for APPS+, we report both pass@1 and pass@10 over all three difficulty categories.

3. Results and Discussion

In this section, we present the results in Table 1, Table 2, and Table 3 and discuss them in detail.

3.1. Offline RL improves code generation capability.

Across both benchmarks and different difficulty levels of APPS+, we observe that offline RL consistently improves the model's code generation performance. The gains are most robust when RLOO is combined with the GRPO-style advantage, underscoring the importance of variance reduction in advantage estimation.

However, we see no improvement for the 7B model on MBPP, suggesting that offline RL offers limited benefits when the base model already performs strongly on a task. Enhancing the offline dataset and incorporating a small amount of online training may be necessary to unlock further gains in such regimes.

3.2. Offline RL improves model on difficult problems.

A key benefit of our offline training setup is improved performance on difficult problems. We observe gains on interview-level tasks for the 0.5B model and on both interview- and competition-level tasks for the 7B model. The effect is more pronounced on competition-level problems with the 7B model: supervised fine-tuning degrades performance on both pass@1 and pass@10, whereas RLOO with GRPO-style advantages improves pass@1 by 88% and pass@10 by 40%.

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.

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

4. Related Works

Code generation is a common application area for LLMs, and many open-source LLMs have been released for this task (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, where feedback is derived from verifiable signals (such as test outcomes) rather than from a learned reward model. However, training is typically carried out in an online fashion.

CodeRL (Le et al., 2022) is an earlier approach that also leverages verifiable rewards but relies on offline data. In CodeRL, candidate solutions are first generated by the model and then used for training, which can be problematic when the base model is weak, as it may fail to produce sufficiently good samples. Moreover, CodeRL employs a frozen critic network for value estimation. In contrast, we train directly on human-written solutions, which ensures the presence of both good and bad examples and yields higher code diversity, and our method does not require a separate critic model.

5. Limitations and Future Work

In this paper, we have shown that offline RL is a promising approach for training LLMs with verifiable rewards for functionally correct code generation. Nonetheless, our study has several limitations:

- We did not systematically explore the space of learning rates, reward formulations, or batch sizes, and in particular relied on relatively small batches.
- The underlying dataset is highly imbalanced; although we applied simple heuristics to mitigate this, more principled data filtering and reweighting strategies could further reduce imbalance.
- Our experiments are purely offline. Combining offline training with a limited amount of online interaction may improve exploration and overall performance.
- We directly apply on-policy online algorithms in an offline setting. Designing theoretically grounded offline RL methods tailored to LLMs, or leveraging value-based approaches such as Q-learning (Snell et al., 2023), could yield further gains.

The above points highlight several promising directions for future work along optimization, data curation, training regimes, and algorithmic design.

6. Conclusion

We have shown that offline RL is an effective and efficient post-training strategy for code-generating LLMs. It improves both pass@1 and pass@10, with pronounced gains on the most challenging problems. Notably, these improvements are achieved under strict computational constraints and with a highly imbalanced dataset.

There remains substantial room for improvement through better reward design and more carefully curated training data. Incorporating additional signals, such as runtime efficiency, memory usage, and security properties, alongside functional correctness could further enhance code quality. Such multi-faceted evaluation is often easier to implement in an offline setting, where candidate solutions can be analyzed along many dimensions without incurring the cost of repeated online execution.

Acknowledgements

This research work was supported by the National Research Center for Applied Cybersecurity ATHENE.

We thank the reviewers for their time and valuable feedback.

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.