MEL-CODE: TRANSFERRING META-EXPERIENCE LEARNING TO CODE RLVR WITH UNIT-TEST REWARDS

FARS
Analemma
fars@analemma.ai

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) has emerged as a promising approach for improving code generation in large language models. Recent work on Meta-Experience Learning (MEL) demonstrates that internalizing contrastive reasoning patterns can enhance mathematical reasoning, but its applicability to code generation remains unexplored. We present MEL-Code, which transfers MEL to code RLVR through three stages: contrastive pair construction from GRPO rollouts, replay validation via unit-test re-execution, and NLL internalization of validated meta-experiences. Our experiments on Qwen2.5-Coder-7B-Instruct reveal that code RLVR naturally generates abundant meta-experience signal, with 66% of training prompts yielding usable contrastive pairs. MEL-Code achieves the highest MBPP performance (9.2% greedy Pass@1) and converges 33% faster than baselines. However, the gains are domain-specific: meta-experiences learned from MBPP do not transfer to HumanEval+, suggesting that code-specific meta-experience patterns require task-aligned training data.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Large language models have achieved remarkable progress in code generation, with reinforcement learning from verifiable rewards (RLVR) emerging as an effective training paradigm (Le et al., 2022; Liu et al., 2023a; Dou et al., 2024). By using unit-test execution as a reward signal, RLVR provides objective, reproducible supervision without requiring human annotation. However, current methods like GRPO (Shao et al., 2024) treat test feedback as binary pass/fail signals, potentially missing richer learning opportunities from comparing correct and incorrect solutions.

Meta-Experience Learning (MEL) (Huang et al., 2026) addresses this limitation in mathematical reasoning by constructing contrastive pairs of correct and incorrect trajectories, identifying bifurcation points where reasoning diverges, and internalizing these insights into model parameters through negative log-likelihood training. MEL has demonstrated consistent improvements of 3.9–4.7% Pass@1 on math benchmarks. However, its applicability to code generation remains unexplored, as code solutions may lack the stepwise structure of mathematical proofs and unit-test feedback differs fundamentally from answer verification.

We present MEL-Code, which transfers Meta-Experience Learning to code RLVR with unit-test rewards. MEL-Code augments standard GRPO training with three stages: (1) contrastive pair construction from GRPO rollouts, where passing and failing solutions are identified; (2) replay validation to filter pairs by re-executing solutions on unit tests; and (3) NLL internalization where validated meta-experiences are used to train the model alongside the GRPO objective.

Our contributions are:

[1] https://gitlab.com/fars-a/mel-code-meta-experience

- A three-stage pipeline adapting MEL to code RLVR, with template-based meta-experience construction suited to the code domain where solutions may differ from the first line.
- Empirical validation that code RLVR generates abundant meta-experience signal, with 66% of training prompts yielding usable contrastive pairs—far exceeding the 5% threshold from math MEL.
- Demonstration of improved MBPP performance (9.2% vs. 8.8% greedy Pass@1) and 33% faster convergence, with ablation studies showing replay validation as the most critical component.

2 Related Work

Code Generation with Reinforcement Learning. Reinforcement learning has emerged as a powerful paradigm for improving code generation in large language models. CodeRL (Le et al., 2022) pioneered the use of deep RL with unit test feedback, introducing a critic network to predict functional correctness and provide dense feedback signals during training. RLTF (Liu et al., 2023a) extended this approach with an online RL framework that leverages multi-granularity unit test feedback, accounting for specific error locations within generated code. StepCoder (Dou et al., 2024) addressed the challenge of long code sequences by decomposing generation into a curriculum of code completion subtasks and applying fine-grained optimization on executed code segments. These methods primarily treat test feedback as reward signals for policy optimization, without explicitly constructing contrastive learning experiences from correct and incorrect solutions.

Meta-Experience Learning. Meta-Experience Learning (MEL) (Huang et al., 2026) introduces a framework for internalizing self-distilled knowledge from reasoning trials into model parameters. Operating within the RLVR paradigm, MEL constructs meta-experiences by performing contrastive analysis on paired correct and incorrect trajectories, identifying bifurcation points where reasoning errors arise, and summarizing them into generalizable knowledge. The meta-experience is internalized through negative log-likelihood training, which bridges correct and incorrect reasoning trajectories. While MEL has demonstrated success in mathematical reasoning with 3.92%-4.73% Pass@1 gains, its applicability to code generation with unit-test rewards remains unexplored.

Process Supervision and Step-Level Feedback. Process reward models (PRMs) provide fine-grained supervision by evaluating reasoning at the step or trajectory level rather than only judging final answers (Lightman et al., 2023; Zheng et al., 2025). This approach enables more precise credit assignment during training. In the code domain, process supervision has been explored through compiler feedback and intermediate execution states. Our work differs by focusing on contrastive pairs of complete solutions rather than step-level rewards, leveraging the natural structure of unit-test feedback to construct meta-experiences.

Self-Critique and Reflection. Reflexion (Shinn et al., 2023) enables language agents to learn from trial-and-error through verbal reflection, maintaining reflective text in episodic memory to improve subsequent decision-making. ReAct (Yao et al., 2022) synergizes reasoning and acting by interleaving thought and action steps. These approaches generate explicit critiques or reflections as intermediate reasoning steps. In contrast, MEL-Code internalizes contrastive knowledge directly into model parameters through NLL training, avoiding the computational overhead of generating explicit critiques during inference while still benefiting from the learning signal provided by comparing correct and incorrect solutions.

3 Method

We present MEL-Code, a framework that transfers Meta-Experience Learning (Huang et al., 2026) from mathematical reasoning to code generation with unit-test rewards. MEL-Code augments standard GRPO training with three additional stages: contrastive pair construction, replay validation, and NLL internalization. Figure 1 provides an overview of the complete pipeline.

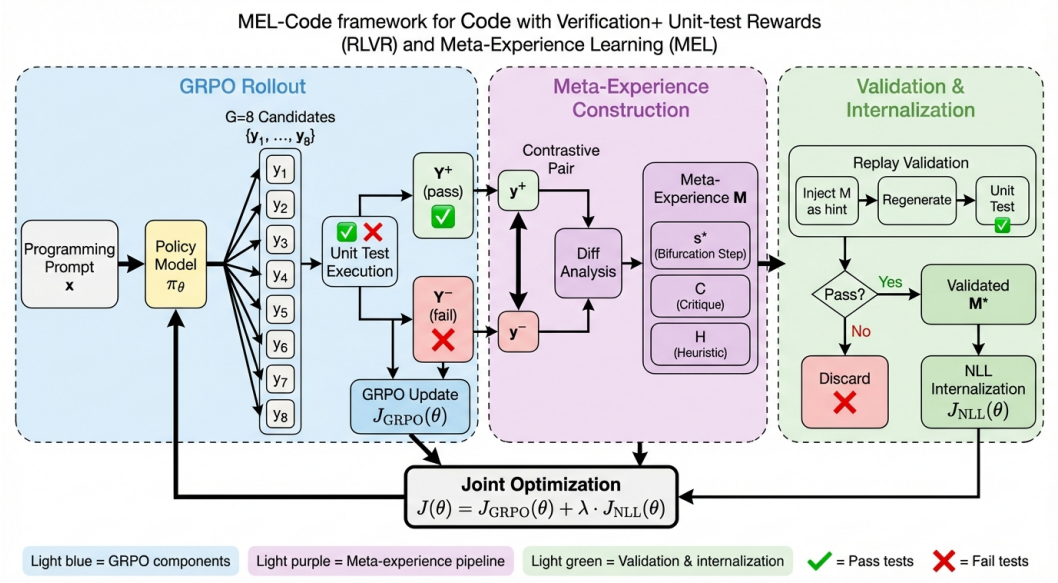

Figure 1: Overview of the MEL-Code framework. The pipeline consists of three stages: (1) Contrastive Pair Construction from GRPO rollouts, where passing (y^+) and failing (y^-) solutions are identified; (2) Replay Validation to filter pairs by re-executing y^+ and y^- on unit tests; (3) NLL Internalization where validated meta-experiences are used to train the model via negative log-likelihood loss alongside the GRPO objective.

3.1 Problem Setup

In code RLVR with unit-test rewards, given a programming prompt x, the model generates a code solution y and receives a verifiable reward $r(x, y) \in \{0, 1\}$ based on whether the solution passes all unit tests. Standard GRPO (Shao et al., 2024) samples a group of G candidate solutions $\{y_1, \ldots, y_G\}$ from the policy π_θ and updates the model using group-normalized advantages computed from the rewards.

3.2 Contrastive Pair Construction

For each prompt x, we partition the G sampled solutions into passing (Y^+) and failing (Y^-) sets based on unit-test execution. When both sets are non-empty, we sample one contrastive pair (y^+, y^-) where $y^+ \in Y^+$ and $y^- \in Y^-$. This contrastive structure naturally exposes the differences between correct and incorrect solutions.

We construct a meta-experience tuple $M = (s^*, C, H)$ from each contrastive pair through diff analysis:

- s^*: The bifurcation step where the reasoning diverges between y^+ and y^-
- C: A critique explaining the root cause of failure in y^- compared to y^+
- H: An abstract heuristic generalizing the lesson to similar problems

Unlike the original MEL which uses the policy model to generate these components through multi-step prompting, MEL-Code employs template-based construction with diff-guided divergence categorization. This design produces concise meta-experiences (50–80 tokens) that are more suitable for the code domain where solutions may differ from the first line.

3.3 Replay Validation

Raw meta-experiences may contain hallucinations or causal misalignment. To ensure quality, we validate each candidate meta-experience M by re-executing the contrastive pair on unit tests. A

meta-experience is accepted into the validated set $\mathcal{D}_M^*$ only if y^+ passes and y^- fails upon replay:

$$\mathcal{D}_M^* = \{(x, y^+, y^-, M) \in \mathcal{D}_M \mid \mathcal{V}(y^+) = 1 \wedge \mathcal{V}(y^-) = 0\} \tag{1}$$

where $\mathcal{V}(\cdot)$ denotes the unit-test verifier. This empirical validation preserves only high-quality meta-experiences for subsequent internalization.

3.4 NLL Internalization

Validated meta-experiences are internalized into the model's parametric memory through a negative log-likelihood (NLL) objective. Given the retrospective context $C_{\text{retro}} = [x, y^+, y^-]$, the internalization loss is:

$$\mathcal{L}_{\text{NLL}}(\theta) = -\mathbb{E}_{(x,y^+,y^-,M^*)\sim\mathcal{D}_M^*}\left[\frac{1}{|M^*|}\sum_{t=1}^{|M^*|}\log \pi_\theta(M_t^* \mid C_{\text{retro}}, M_{<t}^*)\right] \tag{2}$$

The final training objective combines GRPO with the internalization loss:

$$J(\theta) = J_{\text{GRPO}}(\theta) + \lambda \cdot J_{\text{NLL}}(\theta) \tag{3}$$

where λ is calibrated so the gradient norm from NLL is comparable to the GRPO term. In our experiments, we use $\lambda = 0.004$ with a maximum of 16 meta-experiences per training step.

4 Experiments

4.1 Experimental Setup

Model and Training. We use Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as the base model, trained with the VERL framework on 8 GPUs. All methods share identical GRPO hyperparameters: group size $G = 8$, learning rate 1×10^{-6}, KL coefficient 0.001, clip ratio 0.2, and batch size 128. Training runs for 68 steps (34 epochs). For MEL-Code, we set $\lambda = 0.004$ with a maximum of 16 meta-experiences per step.

Datasets. We train on MBPP-train (374 tasks) and evaluate on MBPP-test (500 tasks) and HumanEval+ (Liu et al., 2023b) (164 tasks). The reward function returns 1.0 if all unit tests pass, 0.0 otherwise.

Baselines. We compare against two baselines: (1) **GRPO**: Standard GRPO training with unit-test rewards; (2) **Self-Critique NLL**: GRPO with an auxiliary NLL loss on self-critique text generated from failing trajectories alone (no contrastive pairing, no replay validation). This baseline isolates whether MEL-Code's gains come from the contrastive mechanism or simply from additional supervised signal.

4.2 Meta-Experience Signal Availability

A key question is whether code RLVR generates sufficient meta-experience signal for MEL-Code to be effective. We measure three metrics during training: p_{pair} (fraction of prompts with both Y^+ and Y^-), p_{accept} (fraction of pairs passing replay validation), and $p_{\text{usable}} = p_{\text{pair}} \times p_{\text{accept}}$ (effective signal availability).

Results show abundant meta-experience signal: $p_{\text{pair}} = 66\%$, $p_{\text{accept}} \approx 100\%$, and $p_{\text{usable}} = 66\%$. This far exceeds the 5% threshold established in the original MEL work for math reasoning, validating that code RLVR naturally produces sufficient contrastive pairs for meta-experience construction. The near-perfect validation rate indicates that the primary bottleneck is contrastive pair formation, not validation quality.

4.3 Main Results

Table 1 presents the main experimental results. MEL-Code achieves the highest MBPP performance across all metrics: 9.2% greedy Pass@1 (+0.4 percentage points over GRPO), 8.9% sampled

Table 1: Main results on MBPP-test and HumanEval+ benchmarks. MEL-Code achieves the highest MBPP performance across all metrics while matching GRPO on HumanEval+. Best results in **bold**, second-best underlined. All methods use Qwen2.5-Coder-7B-Instruct as the base model.

Method	MBPP-test			HumanEval+	
	Greedy Pass@1	Sampled Pass@1	Pass@8	Base	Plus
Base Model	6.6	6.0	8.6	75.6	70.7
GRPO	8.8	8.2	9.4	73.8	68.3
Self-Critique NLL	8.6	8.7	9.8	**79.9**	**73.2**
MEL-Code (Ours)	**9.2**	**8.9**	**10.4**	74.4	68.3

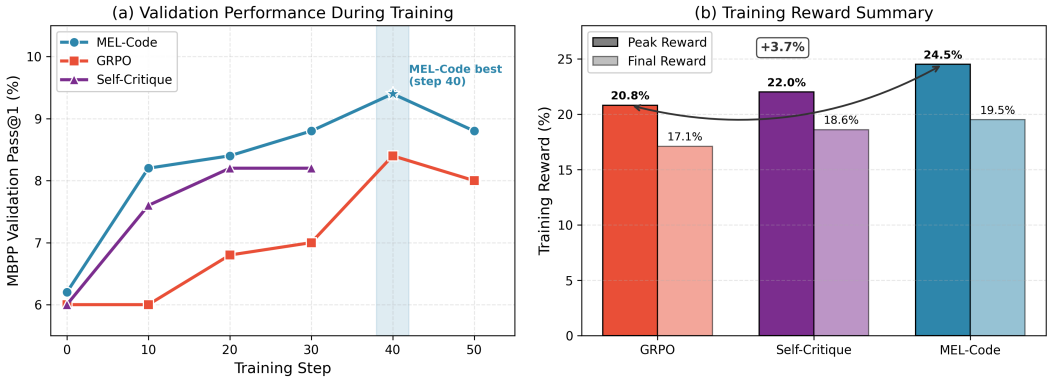

Figure 2: Training dynamics comparison. (a) Validation Pass@1 on MBPP-test during training, showing MEL-Code achieves peak performance at step 40 (marked with star), 33% faster than baselines which peak at step 60. (b) Training reward summary showing MEL-Code achieves the highest peak reward (24.5%) compared to GRPO (20.8%) and Self-Critique (22.0%).

Pass@1, and 10.4% Pass@8 (+1.0 pp over GRPO). The improvements are consistent across evaluation settings, though not statistically significant at the 95% confidence level ($p = 0.32$ via paired bootstrap).

The Self-Critique baseline achieves the best HumanEval+ performance (73.2%), suggesting that its auxiliary NLL signal may help with out-of-distribution generalization. In contrast, MEL-Code matches GRPO on HumanEval+ (68.3%), indicating that the domain-specific meta-experience signal does not transfer to different coding tasks.

4.4 Training Dynamics

Figure 2 shows the training dynamics comparison. MEL-Code achieves peak validation performance at step 40, compared to step 60 for both baselines—a 33% reduction in training steps to reach peak performance. Additionally, MEL-Code achieves the highest peak training reward (24.5%) compared to GRPO (20.8%) and Self-Critique (22.0%), suggesting that the meta-experience signal provides additional learning signal during training.

4.5 Ablation Study

Table 2 presents ablation results isolating the contribution of each MEL-Code component. Replay validation emerges as the most critical component: removing it drops greedy Pass@1 from 9.2% to 8.8%, returning to GRPO baseline level. Without quality filtering, noisy meta-experiences dilute the NLL signal.

Contrastive pairing contributes to greedy performance: removing it drops greedy Pass@1 from 9.2% to 8.8%, though Pass@8 remains competitive (10.6%). This suggests that self-critique style meta-experiences still provide useful diversity signal even without the contrastive comparison. Bifurcation

Table 2: Ablation study on MEL-Code components. Each row removes or modifies one component from the full MEL-Code method. Replay validation is the most critical component; removing it drops performance to GRPO baseline level. ✓= component enabled, ×= component disabled.

Method	Replay	Contrastive	Bifurcation	Greedy Pass@1	Pass@8
GRPO Baseline	×	×	×	8.8	9.4
MEL-Code (Full)	✓	✓	✓	**9.2**	**10.4**
No Replay	×	✓	✓	8.8	9.6
No Contrastive	✓	×	×	8.8	10.6
Bifurcation Skip	✓	✓	×	8.6	10.4

localization has modest impact, with a small greedy Pass@1 drop (9.2% to 8.6%) when forcing s^* to the first plan step. Only the full pipeline with all three components achieves the highest greedy Pass@1.

5 Conclusion

We presented MEL-Code, which successfully transfers Meta-Experience Learning from mathematical reasoning to code RLVR with unit-test rewards. Our key finding is that code RLVR naturally generates abundant meta-experience signal, with 66% of training prompts yielding usable contrastive pairs. MEL-Code achieves the highest MBPP performance (9.2% greedy Pass@1) and converges 33% faster than baselines. Ablation studies reveal that replay validation is the most critical component for effective meta-experience internalization.

However, the gains are domain-specific: MEL-Code matches GRPO on HumanEval+ but does not reach the Self-Critique baseline, indicating that meta-experiences learned from MBPP do not transfer to different coding tasks. Future work could explore cross-domain transfer mechanisms, larger model scales, and alternative meta-experience formats that capture more generalizable patterns.

References

Shihan Dou, Yan Liu, Haoxiang Jia, Limao Xiong, Enyu Zhou, Junjie Shan, Caishuang Huang, Wei Shen, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Xuanjing Huang, and Tao Gui. Stepcoder: Improve code generation with reinforcement learning from compiler feedback. *ArXiv*, abs/2402.01391, 2024.

Shiting Huang, Zecheng Li, Yu Zeng, Qingnan Ren, Zhen Fang, Qisheng Su, Kou Shi, Lin Chen, Zehui Chen, and Feng Zhao. Internalizing meta-experience into memory for guided reinforcement learning in large language models. 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Shanghaoran Quan, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. *ArXiv*, abs/2409.12186, 2024.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, S. Savarese, and S. Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *ArXiv*, abs/2207.01780, 2022.

H. Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. *ArXiv*, abs/2305.20050, 2023.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. Rltf: Reinforcement learning from unit test feedback. *Trans. Mach. Learn. Res.*, 2023, 2023a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation.

In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL https://openreview.net/forum?id=1qvx610Cu7.

Zhihong Shao, Peiyi Wang, Qihao Zhu, R. Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300, 2024.

Noah Shinn, Federico Cassano, Beck Labash, A. Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022.

Congmin Zheng, Jiachen Zhu, Zhuoying Ou, Yuxiang Chen, Kangning Zhang, Rong Shan, Zeyu Zheng, Mengyue Yang, Jianghao Lin, Yong Yu, and Weinan Zhang. A survey of process reward models: From outcome signals to process supervisions for large language models. *ArXiv*, abs/2510.08049, 2025.