Plug-and-Play Data Module for Code RL: Adaptive Ambiguity Replay

Jianqing Zhang[1,2], Wei Xia[2], Zhezheng Hao[3], Hong Wang[2], Hande Dong[2], Qiang Lin[2], Yang Liu[4], Jian Cao[1], Qiang Yang[4]
[1]Shanghai Jiao Tong University, [2]Tencent,
[3]Zhejiang University, [4]Hong Kong Polytechnic University
Correspondence: tsingz@sjtu.edu.cn, xwellxia@tencent.com, cao-jian@sjtu.edu.cn

Abstract

Reinforcement learning (RL) is effective for improving code generation but suffers from data scarcity. While experience replay mitigates this, existing approaches rely on static, in-epoch metrics that overlook training dynamics, often introducing low-utility or outdated data. Analyzing RL dynamics via dataset cartography, we observe that "ambiguous" samples, which are vital for model generalization, rapidly migrate to "easy-to-learn" regions, diminishing their training value. To address this, we propose **Adaptive Ambiguity Replay (A2R)** for RL, a plug-and-play module that prioritizes cross-epoch ambiguous samples. To neutralize the noise from stale experiences, **A2R** incorporates an adaptive importance mechanism based on policy divergence to weigh replayed rollouts. Extensive experiments on nine LLMs (3B–14B) demonstrate that **A2R** outperforms state-of-the-art baselines on real-world code editing tasks across both unseen and learned domains. Our results highlight cross-epoch ambiguity as a key factor for effective replay in RL. Code: https://github.com/TsingZ0/verl-A2R.

1 Introduction

In recent years, code models such as large language models (LLMs) (Dong et al., 2025) have achieved increasing success across a wide range of software engineering tasks (Gao et al., 2025). Code editing is among the most critical capabilities of these models, as nearly all code agents interact with their environments through editing actions (Guo et al., 2024b; Cassano et al., 2024). In real-world code editing scenarios, the context is often complex, encompassing edit histories, code ranges, cursor positions, and user intent, among other factors (Zhang et al., 2025b; Yang et al., 2024).

Reinforcement learning (RL) is among the most effective approaches for improving coding capabilities, but it suffers from the scarcity of high-quality training data (Wang et al., 2024). Replay mechanisms alleviate this issue by reusing high-quality historical prompts and rollouts, thereby maximizing the value extracted from limited data (Chen et al., 2024; Butt et al., 2024). A replay mechanism involves three key design questions: (1) *what* to replay (experience selection), (2) *where* to replay (integration with main training or not), and (3) *how* to replay (implementation details).

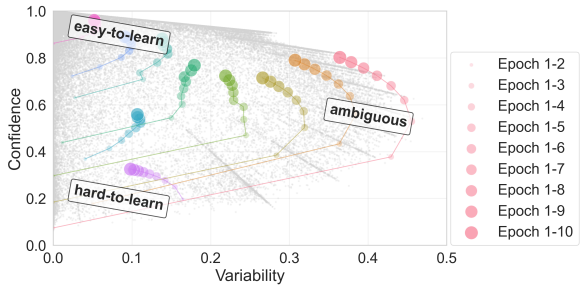

Figure 1: Based on model *training dynamics*, we tracked the *trajectories* of randomly picked 10 data points across three regions in the dataset cartography (Swayamdipta et al., 2020). Here, confidence (y-axis) and variability (x-axis) represent the mean and standard deviation of rewards tracked from epoch 1 to the current epoch. Gray points represent the dataset cartography computed from rewards across epochs 1-10.

Among these three aspects, *what* to replay is the most critical factor distinguishing replay methods (Pleiss et al., 2025), as the quality of replayed experiences directly impacts learning. While replay has been extensively studied in classical RL (Vanseijen and Sutton, 2015), relatively few works address replay in LLM-based RL. Existing LLM RL approaches, such as RePO (Li et al., 2025) and RLEP (Zhang et al., 2025a), primarily select replay experiences using in-epoch metrics, including variance and reward. These metrics are inherently static and fail to capture training dynamics as the model evolves, which limits generalization, particularly when fine-tuning on new domains. Moreover, in-

Findings of the Association for Computational Linguistics: ACL 2026, pages 17865–17875
July 2-7, 2026

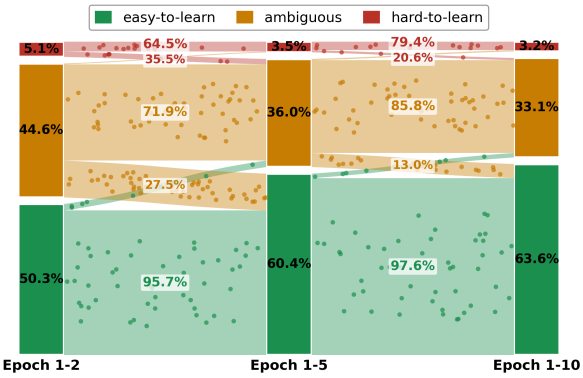

Figure 2: The *migration from left to right* of data points across the three regions during training.

tegrating *outdated* replay experiences with current training steps, as in RePO, can confuse the model (Zheng et al., 2025).

To address these issues, we propose a *training-dynamics-aware*, *plug-and-play* data module called Adaptive Ambiguity Replay (**A2R**) for RL, which operates independently of the main training. **A2R** is motivated by our observations of cross-epoch data dynamics. In Figure 1, we capture cross-epoch training dynamics by tracking the migration trajectories of data points. Over several epochs, most points shift from the "hard-to-learn" or "ambiguous" regions toward the "easy-to-learn" region. While the majority move toward easier regions, a few migrate back (see Figure 2), causing the ambiguous region to shrink rapidly. As noted in dataset cartography (Swayamdipta et al., 2020), cross-epoch ambiguous data enhance generalization to unseen domains, whereas easy-to-learn data ensure convergence.

Motivated by these observations, our **A2R** replays ambiguous data to maximize data utility. Since dataset cartography is computed after model finishes updating and the model continues to evolve, we estimate each data point's position in the cartography and select the top-K most ambiguous points and their rollouts for replay during training. For outdated rollouts from the old policy, we adaptively compute the replay sample importance α based on the divergence between the current and old policies. Larger divergence results in a lower α. This approach prioritizes ambiguous data that aligns with the direction of policy updates.

We integrate **A2R** into popular LLM RL methods, GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025), and evaluate them on nine general-purpose and code-specialized LLMs ranging from 3B to 14B parameters. For code editing tasks, we use real-world data with complex contexts. The results demonstrate the effectiveness of **A2R** across extensive experiments on unseen and learned domains. Overall, our contributions are as follows:

- We are the first to adapt the dataset diagnosis tool, dataset cartography, for RL dynamics analysis, revealing data migration and ambiguous data loss.
- We propose a plug-and-play data module, **A2R**, that maximizes the utility of ambiguous data for improved generalization and adaptively evaluates data importance using policy divergence on stale rollouts.
- We show the effectiveness of **A2R** across nine general-purpose and code-specialized LLMs, spanning 3B to 14B parameters, on real-world code editing tasks in both in-domain and out-of-domain settings.

2 Related Work

2.1 Experience Replay for LLM RL

Experience replay is widely used in LLM RL to improve sample efficiency by reusing previously generated rollouts. One line of work focuses on replaying successful trajectories. For example, RLEP (Zhang et al., 2025a) mixes verified-correct rollouts from a success buffer with fresh on-policy data, and similar positive replay buffers are adopted in code RL (Sun et al., 2025b). Another line integrates replay directly into the optimization objective. RePO (Li et al., 2025) augments the on-policy objective with an off-policy replay loss over buffered rollouts and prioritizes replay using simple heuristics such as return variance. Other work increases replay frequency (*e.g.*, LoRR (Liu et al., 2025)) or relies on task-specific signals (Dou et al., 2025; Wang et al., 2025b; Chen et al., 2024; Sun et al., 2025a; Ravindran, 2025). However, most approaches use static, in-epoch criteria and do not track how experience utility evolves during training, leading to inefficient or stale replay.

2.2 In-Epoch Ambiguity in LLM RL

Recent studies on LLM RL identify ambiguity as a primary source of effective learning signal: for a fixed prompt, different rollouts may diverge, revealing policy uncertainty. One line of work exploits this ambiguity through variance-based data selection. DAPO (Yu et al., 2025) discards prompts with

degenerate outcomes (all-correct or all-incorrect) and concentrates training on prompts exhibiting mixed rewards, while PODS (Wang et al., 2025a) retains only high-variance rollouts for policy updates. Beyond selection, a closely related line leverages ambiguity for sample reweighting during training. Re-Schedule (Wang et al., 2025c) and DOTS (Sun et al., 2025a) assign higher priority to queries with greater rollout ambiguity, whereas SEED-GRPO (Chen et al., 2025) quantifies ambiguity via semantic entropy over sampled answers and uses this signal to modulate GRPO updates. However, most existing ambiguity-based RL methods rely on *in-epoch* measurements, overlooking the dynamic evolution of ambiguity as the model progresses through training.

3 Preliminaries

3.1 Problem Formulation

We formulate the code editing task as a conditional sequence generation problem (Chen et al., 2021; Zhuo et al., 2024). Let x denote the input context, which encompasses the repository snapshot, edit history, and cursor position (Zhang et al., 2023), and let y^* represent the ground-truth edit. The objective is to learn a policy π_θ that generates a predicted code patch $\hat{y}$ to maximize the expected reward $\mathbb{E}_{\hat{y}\sim\pi_\theta}[r(\hat{y}, y^*)]$, where the reward function $r(\cdot)$ measures the functional and structural alignment with the user's intent (Le et al., 2022).

3.2 RL for Code Editing

We adopt the RL framework for post-training, where the LLM functions as a stochastic policy $\pi_\theta(a|s)$ (Ouyang et al., 2022). Here, the state s corresponds to the input prompt x, and the action a consists of the generated tokens forming $\hat{y}$. A trajectory is defined as $\tau = (s, a)$, representing a complete sampling sequence. The optimization generally minimizes a loss function $\mathcal{L}(\theta)$ that incorporates an advantage-weighted objective and a KL-divergence constraint to ensure training stability (Schulman et al., 2017; Shao et al., 2024):

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[-\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A} + \beta\mathbb{D}_{\text{KL}}(\pi_\theta||\pi_{\text{ref}})\right], \quad (1)$$

where $\hat{A}$ denotes the estimated advantage value, $\pi_{\theta_{\text{old}}}$ represents the sampling policy, and π_{ref} is the reference model. We present only the generic RL formulation here, as **A2R** is plug-and-play and can be readily integrated into existing frameworks.

4 Method

4.1 Observation and Motivation

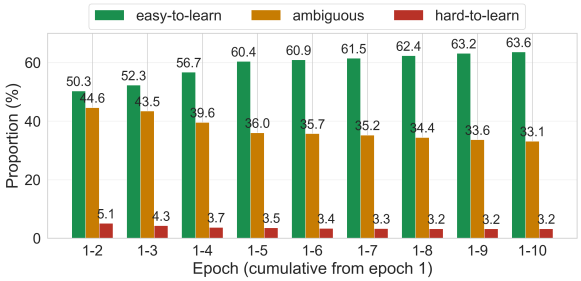

Figure 3: Dynamic changes in data proportions across three regions in the dataset cartography during model training, observed here on Qwen2.5-Coder-7B (Hui et al., 2024) using DAPO (Yu et al., 2025).

In this work, we propose the plug-and-play data module **A2R** for RL, motivated by observations of training dynamics in ambiguous data utility across epochs. As shown in Figure 1, most data points tend to move from lower to higher rewards, with ambiguity initially increasing and then decreasing as rewards rise. In other words, their utility evolves from the "easy-to-learn" region to the "ambiguous" region and eventually to the "hard-to-learn" region. Beyond these trends, we also observe migration patterns during training, as shown in Figure 2, where most points move from the "ambiguous" region to the "easy-to-learn" region, with few moving back, which causes the "ambiguous" region to shrink quickly. Prior work (Swayamdipta et al., 2020) shows that "ambiguous" data can substantially improve out-of-domain (OOD) performance. Accordingly, we propose replaying ambiguous data together with their corresponding rollouts to enhance data utility and improve model performance on unseen domains.

Figure 3 further illustrates the *dynamic* changes in data proportions across the three regions, with most migration occurring in the early epochs. As training progresses, the migration pattern evolves dynamically. Therefore, it is necessary to *adaptively* track the migration at each training step and select the currently ambiguous data to maximize their utility for model updates, a process we refer to as *adaptive ambiguity replay (***A2R***)*.

4.2 Adaptive Ambiguity Replay

Most existing replay methods adopt invasive designs that modify rollouts, advantage computation, grouping, *etc*., in the original RL algorithms (Li

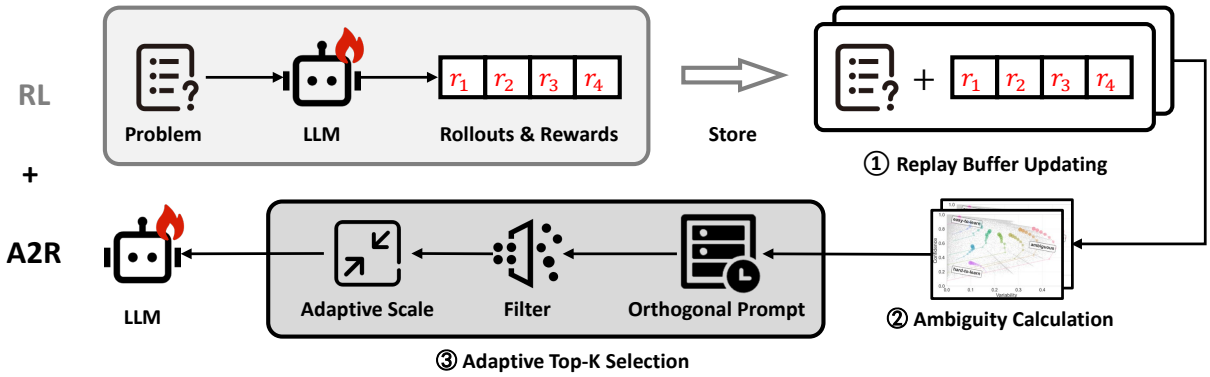

Figure 4: Illustration of **A2R** integrated with generic RL methods.

et al., 2025; Zhang et al., 2025a), which limits them to specific classes of RL methods. In contrast, our plug-and-play **A2R** functions as a plug-in and can be directly applied to generic RL methods. As illustrated in Figure 4, our **A2R** consists of three components: (1) replay buffer updating, (2) ambiguity calculation, and (3) adaptive top-K selection.

4.2.1 Replay Buffer Updating

The replay buffer $\mathcal{R} = (\mathcal{M}, \mathcal{T})$ consists of two mappings: the metric mapping $\mathcal{M} : p \mapsto \{r_1, r_2, \ldots, r_E\}$ stores reward values for each prompt p across E epochs, and the batch mapping $\mathcal{T} : p \mapsto \{\tau_1, \tau_2, \ldots, \tau_n\}$ stores n rollout trajectories along with necessary information, such as the corresponding logarithm probabilities. To balance ambiguity accuracy and resource cost, we dynamically store rollouts from each batch and update the replay buffer at the end of each epoch. During this update, we apply a first-in-first-out (FIFO) mechanism (Isele and Cosgun, 2018) to retain the most recent rollouts, providing an efficient ambiguity estimation without additional computation overhead.

4.2.2 Ambiguity Calculation

We calculate ambiguity only use $\mathcal{M}$. Inspired by dataset cartography (Swayamdipta et al., 2020), we define the training dynamics of the sample with prompt p, as statistics computed across E epochs in the RL setting. These statistics then serve as coordinates (**confidence** and **variability**) on the dataset cartography map. The **confidence** is defined by the mean of the recorded rewards across epochs:

$$\hat{\mu}_p^E = \frac{1}{E} \sum_{e=1}^{E} r_e, \tag{2}$$

Besides, the **variability** is defined as the standard deviation of the rewards across epochs:

$$\hat{\sigma}_p^E = \sqrt{\frac{\sum_{e=1}^{E} (r_e - \hat{\mu}_p^E)^2}{E}}. \tag{3}$$

A sample on which the model remains indecisive throughout training will exhibit high variability. This sample with prompt p is thus assigned coordinates $(\hat{\sigma}_p^E, \hat{\mu}_p^E)$, indicating its position on the map after the model completes E training epochs.

4.2.3 Adaptive Top-K Selection

Before selection, we exclude prompts from the replay buffer that also appear in the current batch, which preserves the plug-and-play property of **A2R** by avoiding explicit update conflicts with the original RL procedure. Thus, we obtain an orthogonal prompt set $\mathcal{P}$.

Then, we perform top-K selection on $\mathcal{P}$ by filtering out samples with low ambiguity in the dataset cartography. At each training step, we select the top-K prompts into $\mathcal{P}_K$ and their rollouts with the largest $\hat{\sigma}$ values, as $\hat{\sigma}$ alone is sufficient to distinguish the ambiguous region from the other two regions.

Given the prompts in $\mathcal{P}_K$, we retrieve the corresponding past rollouts $\mathcal{T}' \leftarrow \bigcup_{p \in \mathcal{P}_K} \mathcal{T}[p]$. However, treating all replay samples equally can be suboptimal, as outdated experiences may adversely impact current policy updates, a common limitation of vanilla replay approaches (Fedus et al., 2020). To address this, we adaptively adjust the importance of the selected replay samples by measuring the policy divergence δ_j for the j-th mini-batch on

Algorithm 1 Adaptive Ambiguity Replay (**A2R**)

for epoch $e = 1$ to E **do**
  // Phase 1: Load historical replay buffer
  if $e > 1$ **then**
    Load $\{\mathcal{M}^{(i)}\}_{i=1}^{e-1}$ and $\mathcal{T}^{(e-1)}$
  Initialize: $\mathcal{M}^{(e)} \leftarrow \{\}$, $\mathcal{T}^{(e)} \leftarrow \{\}$
  for all batch $\mathcal{B}$ **do**
    // Phase 2: Standard Policy Update
    Generate rollouts and compute rewards.
    Update θ using the original loss $\mathcal{L}(\theta)$.
    // Phase 3: Update Replay Buffer
    for all p with trajectories $\{\tau_i\}_{i=1}^{n}$ **do**
      $\mathcal{M}^{(e)}[p] \leftarrow \frac{1}{n}\sum_{i=1}^{n} r(\tau_i)$
      $\mathcal{T}^{(e)}[p] \leftarrow \{\tau_i\}_{i=1}^{n}$
    // Phase 4: Adaptive Replay
    if $e > 1$ **then**
      $\mathcal{P} \leftarrow \{p : p \in \bigcup_{i=1}^{e-1} \mathcal{M}^{(i)} \wedge p \notin \mathcal{B}\}$
      Get σ_p based on dataset cartography.
      $\mathcal{P}_K \leftarrow \text{argtop}_K\{\sigma_p : p \in \mathcal{P}, \sigma_p > 0\}$
      $\mathcal{P}_K \leftarrow \mathcal{P}_K \cap \text{keys}(\mathcal{T}^{(e-1)})$
      if $|\mathcal{P}_K| > 0$ **then**
        $\mathcal{T}' \leftarrow \bigcup_{p \in \mathcal{P}_K} \mathcal{T}^{(e-1)}[p]$
        // Adaptive Sample Importance
        for all mini-batch $\mathcal{T}'_j \subseteq \mathcal{T}'$ **do**
          Get α_j using Eq. (4) and Eq. (5).
          $\mathcal{L}_{\text{scaled}}(\theta) \leftarrow \alpha_j \cdot \mathcal{L}(\theta)$
          Update θ with $\mathcal{L}_{\text{scaled}}(\theta)$
  // Phase 5: Persist replay buffer
  Save $\mathcal{R}^{(e)} = (\mathcal{M}^{(e)}, \mathcal{T}^{(e)})$
return π_θ

$\mathcal{T}'_j \subseteq \mathcal{T}'$:

$$\delta_j = \frac{1}{|\mathcal{T}'_j|} \sum_{\tau \in \mathcal{T}'_j} \left| \log \pi_\theta(a|s) - \log \pi_{\theta_{\text{prev}}}(a|s) \right|, \tag{4}$$

where π_θ is the current policy model and $\pi_{\theta_{\text{prev}}}$ is the previous policy model that generated the old rollouts. The log probabilities under $\pi_{\theta_{\text{prev}}}$ are already stored in $\mathcal{T}'_j$, incurring no additional computation. We define the adaptive sample importance $\alpha_j \in (0, 1]$ to be

$$\alpha_j = \exp(-\lambda \cdot \delta_j), \quad \lambda > 0, \tag{5}$$

where λ controls the intensity. We scale the loss $\mathcal{L}(\theta)$ by α_j. Greater divergence indicates lower importance, thereby reducing the influence of overly outdated rollouts. To align with existing batch-level backpropagation, we measure importance at the mini-batch level. A detailed version of the overall algorithm is presented in algorithm 1.

5 Experiment

Related Baselines. (1) RePO (Li et al., 2025) mixes online and offline rollouts from a replay buffer to balance online and offline RL, using various strategies to select replay samples. We adopt the most closely related in-epoch variance-driven strategy. (2) RLEP (Zhang et al., 2025a) targets hard problems by storing successful rollouts in a replay buffer and replaying them to provide more correct answers in subsequent training. RLEP is originally designed for discrete-reward settings, such as mathematical reasoning, and is *not well suited to continuous-reward scenarios*, where finding "successful" samples is nontrivial. EFRame (Chen et al., 2024) is identical to RLEP in its replay mechanism. (3) GRPO serves as the base RL algorithm widely used in both industry and academia. (4) DAPO is a practical GRPO variant that improves data utility through dynamic sampling and higher clipping ratios. The baselines RePO and RLEP are implemented on top of DAPO.

Training and Evaluation Data. In this study, we focus on *code edit in real-world software engineering*, where each user maintains complex contextual codebases accompanied by multiple editing histories. These histories capture the users' individual coding and editing preferences. After an extensive search of publicly available benchmarks, we found only zeta (Zed Industries, 2025) that captures realistic code-editing behaviors, but it contains only a few hundred examples. Therefore, we collected a large-scale dataset of over 50,000 practical code-editing instances from internal users, reflecting real daily software development activities. All data containing sensitive or private content was identified and removed. Then, we evaluate the model on both the internal test set (3,897 cases in total) and zeta.

The statistics of the training dataset are illustrated in Figure 5. The dataset comprises a total of 51,844 code-editing tasks spanning 10 distinct programming languages. The three most represented languages are Go (37.71%), Python (22.14%), and Java (21.03%), which together account for 80.88% of all samples. The remaining languages, C++, Kotlin, TypeScript, JavaScript, C, Rust, and Lua, appear less frequently, with Rust and Lua being the least represented (0.43% and 0.12%, respectively).

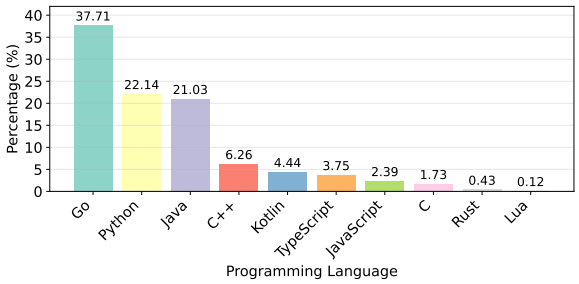

Figure 5: Language distribution statistics of the training dataset for real-world code edit.

Large Language Models. For a comprehensive evaluation, we examine **nine** large language models (LLMs) that span both general-purpose (Mistral-v0.3-7B (Jiang et al., 2023), Qwen2.5-3B/7B (Qwen Team, 2024), Qwen3-4B/8B (Yang et al., 2025)) and code-specialized (Qwen2.5-Coder-3B/7B/14B (Hui et al., 2024), DeepSeek-Coder-6.7B (Guo et al., 2024a)) settings. We use their instruction-tuned versions obtained from Hugging Face (Hugging Face, 2025).

Implementation. We implement our plug-and-play **A2R** on top of the widely used RL framework `verl` (Sheng et al., 2025), keeping the existing training pipeline unchanged and adding only a few code snippets to realize **A2R**. Integrating **A2R** into RL algorithms is straightforward, functioning more like a plug-in compared to other methods (*e.g.*, RePO and RLEP) that require substantial modifications to the original training scheme. We follow the default training hyperparameters of the target RL algorithm when adding the replay feature. For the reward model, we follow prior code-editing work (Zhang et al., 2025b) and combine the exact match (em) (Dibia et al., 2023) and normalized edit distance (ed^*) metrics: $r(\hat{y}, y^*) := \frac{1}{2}[em(\hat{y}, y^*) + ed^*(\hat{y}, y^*)]$, where $\hat{y}$ is the predicted code snippet and y^* is the corresponding ground truth. The normalized edit distance is defined as

$$ed^*(y_a, y_b) := 1 - \frac{ed(y_a, y_b)}{\max\{len(y_a), len(y_b)\}},$$

where $ed(y_a, y_b)$ is the plain edit distance computed via the dynamic programming algorithm (Lcvenshtcin, 1966) and $len(\cdot)$ denotes the length of a code snippet. Following prior work (Deng et al., 2025), we evaluate performance using the exact match (em) metric. Additional details are provided in the Appendix.

Table 1: Exact match accuracy on the zeta "dpo" set for Qwen2.5-Coder and Qwen3. The asterisk (*) denotes reasoning LLMs.

Name	Qwen3*		Qwen2.5-Coder	
Size	4B	8B	3B	7B
RePO	17.42	18.18	15.15	15.91
RLEP	8.33	10.61	11.36	12.88
DAPO	12.88	19.70	13.64	16.67
+ **A2R**	18.18	18.94	15.15	22.73
GRPO	8.33	13.64	17.42	21.97
+ **A2R**	10.61	16.67	19.70	25.00

Table 2: Exact match accuracy on the *tiny* zeta "test" set for Qwen2.5-Coder and Qwen3.

Model	Qwen3*		Qwen2.5-Coder	
Size	4B	8B	3B	7B
RePO	21.21	30.30	24.24	30.30
RLEP	18.18	24.24	18.18	21.21
DAPO	21.21	33.33	27.27	30.30
+ **A2R**	24.24	36.36	30.30	33.33
GRPO	21.88	21.88	34.38	37.50
+ **A2R**	25.00	28.12	40.62	43.75

5.1 Generalization to Unseen Domains

Different from our collected training data, the open-source zeta dataset covers unseen domains. We evaluate our **A2R** on zeta's two subsets "dpo" and "eval", as shown in Table 1 and Table 2. Note that both "dpo" and "eval" set are small, leading to stepped exact match accuracy. Though small in scale, zeta stands out as the only publicly available open-source dataset tailored for real-world code editing tasks. Overall, our **A2R** consistently outperforms RLEP and RePO, achieving up to a 6.82-point improvement by enhancing DAPO by 6.06 points, corresponding to a relative gain of 36.36%.

The core idea of RLEP is to store successful rollouts in a replay buffer under discrete-reward settings. However, this mechanism does not transfer well to continuous-reward code editing scenarios, where finding completely successful rollouts is non-trivial (see Sec. A for details), leading to degraded performance. Moreover, RLEP does not update replayed experiences, which becomes a significant limitation in large-scale training with many optimization steps. As the policy evolves, stale ex-

Table 3: Exact match accuracy on the learned-domain test set using nine LLMs.

Model	Mistral-v0.3	Qwen2.5		Qwen3*		Qwen2.5-Coder			DeepSeek-Coder
Size	7B	3B	7B	4B	8B	3B	7B	14B	6.7B
RePO	16.86	33.67	39.49	38.51	40.18	40.07	41.88	—	39.89
RLEP	11.37	20.14	22.90	24.11	25.43	24.81	27.31	—	26.77
DAPO	16.59	32.80	39.46	37.99	39.80	38.74	41.64	—	41.09
+ **A2R**	16.90	35.84	40.07	40.31	42.91	42.18	43.36	—	42.16
Improve.	**+0.31**	**+3.04**	**+0.61**	**+2.32**	**+3.11**	**+3.44**	**+1.72**	—	**+1.07**
GRPO	12.93	38.80	39.05	39.47	36.80	39.69	40.05	42.64	23.32
+ **A2R**	13.87	39.76	39.89	40.52	37.92	40.54	41.16	43.89	23.86
Improve.	**+0.94**	**+0.96**	**+0.84**	**+1.05**	**+1.12**	**+0.85**	**+1.11**	**+1.25**	**+0.54**

periences become increasingly misaligned with the current model, resulting in a substantial negative impact on performance.

On the contrary, RePO dynamically updates the replay buffer alongside model training, similar to our **A2R**, which enables RePO to achieve strong performance. However, RePO selects replay samples solely based on in-epoch metrics, without accounting for training dynamics across epochs or the direction of model updates, which limits its effectiveness.

Beyond replay buffer updating and adaptive policy divergence control, our **A2R** further selects top-K samples exhibiting cross-epoch ambiguity, thereby providing additional performance gains, particularly on unseen domains. Note that RePO samples an identical number of instances (512) per rollout batch, whereas our **A2R** method only selects $K = 128$ samples, thus incurring merely $1/4$ of the overhead required by RePO.

5.2 Performance on the Learned Domain

In addition to out-of-domain (OOD) performance, we show in Table 3 that **A2R** also improves in-domain performance on a collected larger test set that follows the same distribution as the training data. Here, we also include a broader set of models with relatively weaker performance on code editing tasks, such as Mistral-v0.3, *etc*. In our long-context setting, DAPO hits out-of-memory (OOM) errors with 14B models, causing missing results.

We observe that our adaptive ambiguity replay module provides greater improvements for originally strong models, such as the Qwen3 and Qwen2.5-Coder series. This is expected, as strong models generate higher-quality experiences and more valuable old rollouts for replay, whereas experiences from weaker models are more likely to hinder the updating model's progress. In the learned domain, which emphasizes specificity over generalization, the improvement over GRPO is smaller than over DAPO. This is because GRPO does not dynamically sample high-variance instances, leading to more data per epoch and longer training steps. Longer epochs mean older rollouts and greater policy divergence during replay, which limits the effectiveness of our **A2R**. We address this limitation by replaying only the prompts and regenerating the rollouts (see Sec. 5.4).

5.3 Performance Curves

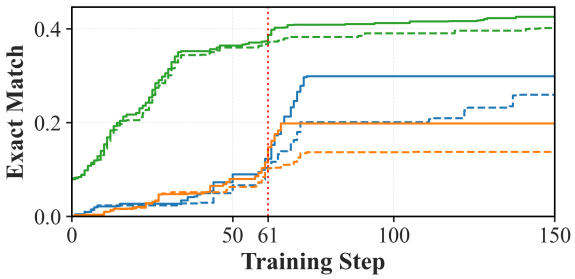

Figure 6: Exact-match curves on the collected test set (green), zeta test set (blue), and zeta dpo set (orange) using Qwen2.5-Coder-7B. Solid lines denote DAPO+**A2R**, while dotted lines denote DAPO.

We present the stepped performance curves in Figure 6. For DAPO, **A2R** collects rollouts up to the 61st step and begins replay thereafter. The curves show a clear performance gain once **A2R** is enabled, elevating model capability, particularly on the unseen zeta domain. Moreover, **A2R** achieves stronger performance at earlier stages, creating additional opportunities for early stopping to reduce time and computational cost while attaining better results.

Table 4: Exact match accuracy on the learned-domain test set using different hyperparameters (K and λ) on Qwen2.5-Coder-3B. DAPO achieves 38.74 here.

	K					λ				
Values	32	64	128	256	512	0.5	1	2	5	10
DAPO + **A2R**	39.59	40.62	42.18	42.22	**42.25**	38.33	40.19	41.64	**42.18**	41.71

5.4 Regenerating Rollouts

Table 5: Exact match accuracy on the learned-domain test set using Qwen2.5-Coder and Qwen3 with regenerated rollouts.

Model	Qwen3*		Qwen2.5-Coder	
Size	4B	8B	3B	7B
DAPO + **A2R**	40.31	42.91	42.18	43.36
+ **A2R** (regen)	40.76	43.44	42.82	43.90
Improve.	**+0.45**	**+0.53**	**+0.64**	**+0.54**
GRPO + **A2R**	40.52	37.92	40.54	41.16
+ **A2R** (regen)	40.98	39.75	42.03	43.28
Improve.	**+0.46**	**+1.83**	**+1.49**	**+2.12**

Besides replaying both prompts and their corresponding old rollouts, we propose a variant that replays only the ambiguous prompts and regenerates rollouts when the prompts are selected. This is reasonable because the confidence and variance used in dataset cartography depend only on the prompts, not on the rollouts.We previously retained old rollouts to reduce computational cost. Regenerating rollouts instead uses an up-to-date policy, which reduces policy divergence and mitigates the negative effects of standard replay. As shown in Table 5, this variant yields consistent improvements, particularly for GRPO-based methods, with gains of up to 2.12 points. This is because the step interval between epochs is too large, causing the old rollouts stored in the replay buffer to become overly outdated.

5.5 Hyperparameter Study

Our **A2R** introduces two hyperparameters, K and λ, which we tune using DAPO + **A2R**. The parameter K controls the number of replayed prompts: increasing K replays more samples at higher computational cost, while decreasing K reduces overhead. Although slightly higher exact match accuracy can be achieved with $K > 128$, we find that $K = 128$ offers a good balance between performance and efficiency. The parameter λ controls the strength of adaptive importance scaling; its effect follows a unimodal (parabolic) trend, with $\lambda = 5$ yielding the best performance and thus used as the default.

5.6 Ablation Study

Table 6: Exact match accuracy on the learned-domain test set using Qwen2.5-Coder and Qwen3 for ablation study. "OrtPrompt" and "AdaScale" are short for the orthogonal prompt and adaptive scale, respectively. σ denotes the variance.

Model	Qwen3*		Qwen2.5-Coder	
Size	4B	8B	3B	7B
DAPO	37.99	39.80	38.74	41.64
DAPO + **A2R**	40.31	42.91	42.18	43.36
* Reward (↑)	36.96	38.27	37.02	39.77
* Reward (↓)	30.39	31.53	34.61	38.83
* In-Epoch σ (↑)	35.61	38.46	37.71	40.19
- OrtPrompt	37.20	39.11	37.63	40.38
- Filter	38.29	39.89	38.94	41.88
- AdaScale	38.87	41.35	40.19	42.28

Here we study the effectiveness of the ambiguity calculation and adaptive top-K selection components in **A2R**. From Table 6, we observe that the primary gains of **A2R** stem from cross-epoch ambiguous data replay. In contrast, using alternative criteria for top-K selection, such as reward (↑), reward (↓), or in-epoch variance σ (↑), degrades training and yields worse performance than DAPO. Among these, the reward (↓) criterion performs the worst, indicating that replaying hard samples is detrimental. Taking a closer look at the adaptive top-K selection, we find that using non-orthogonal prompts in the current training step can introduce conflicts and degrade accuracy. Even random top-K selection (–Filter) yields modest gains once the orthogonal prompt operation is applied. Incorporating adaptive scaling (AdaScale) further improves **A2R**, while removing AdaScale still allows **A2R** to consistently outperform DAPO.

6 Conclusion

This work introduces **A2R**, a plug-and-play module that enhances RL for code editing by leveraging training dynamics. By replaying samples with high cross-epoch ambiguity and using adaptive importance, **A2R** prioritizes high-utility data while mitigating policy divergence. Experiments on nine LLMs demonstrate gains in performance and generalization, highlighting the benefits of leveraging ambiguous data in LLM RL.

7 Limitations

The benefit of replaying historical data decreases as the model diverges from the version that generated it. When the interval between epochs is too large, cached rollouts can become outdated, making naive replay ineffective. This is a common limitation of replay-based methods. We show that recomputing rollouts through a regeneration strategy can effectively address this issue, albeit with additional computation.

Acknowledgments

This research is supported by Voicecomm. It is also supported by the Smart Medical Special Research Fund of the Shanghai Municipal Health Commission (Grant No. 2025ZHYL003) and the Interdisciplinary Program of Shanghai Jiao Tong University (project number YG2024QNB05). Besides, this work is also supported by the Presidential Young Scholar Scheme (Project No. P0056638) and the RIAIoT (Project No. P0059914) at The Hong Kong Polytechnic University. We also thank Tencent CodeBuddy (https://www.codebuddy.ai/) and WeChat Game team for their valuable support.

References

Natasha Butt, Blazej Manczak, Auke Wiggers, Corrado Rainone, David W Zhang, Michaël Defferrard, and Taco Cohen. 2024. Codeit: Self-improving language models with prioritized hindsight replay. In *International Conference on Machine Learning*.

Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, and 1 others. 2024. Can it edit? evaluating the ability of large language models to follow code editing instructions. In *First Conference on Language Modeling*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Minghan Chen, Guikun Chen, Wenguan Wang, and Yi Yang. 2025. Seed-grpo: Semantic entropy enhanced grpo for uncertainty-aware policy optimization. *arXiv preprint arXiv:2505.12346*.

Yuyang Chen, Kaiyan Zhao, Yiming Wang, Ming Yang, Jian Zhang, and Xiaoguang Niu. 2024. Enhancing llm agents for code generation with possibility and pass-rate prioritized experience replay. *arXiv preprint arXiv:2410.12236*.

Le Deng, Xiaoxue Ren, Chao Ni, Ming Liang, David Lo, and Zhongxin Liu. 2025. Enhancing project-specific code completion by inferring internal api information. *IEEE Transactions on Software Engineering*.

Victor Dibia, Adam Fourney, Gagan Bansal, Forough Poursabzi-Sangdeh, Han Liu, and Saleema Amershi. 2023. Aligning offline metrics and human judgments of value for code generation models. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*.

Shihan Dou, Muling Wu, Jingwen Xu, Rui Zheng, Tao Gui, and Qi Zhang. 2025. Improving rl exploration for llm reasoning through retrospective replay. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 594–606. Springer.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. 2020. Revisiting fundamentals of experience replay. In *International conference on machine learning*.

Cuiyun Gao, Xing Hu, Shan Gao, Xin Xia, and Zhi Jin. 2025. The current challenges of software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024a. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, and 1 others. 2024b. Codeeditorbench: Evaluating code editing capability of large language models. *arXiv preprint arXiv:2404.03543*.

Hugging Face. 2025. Hugging face — the ai community building the future. https://huggingface.co/. Accessed: 2025-10-08.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

David Isele and Akansel Cosgun. 2018. Selective experience replay for lifelong learning. In *Proceedings of the AAAI conference on artificial intelligence*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

VI Lcvenshtcin. 1966. Binary coors capable or 'correcting deletions, insertions, and reversals. In *Soviet physics-doklady*.

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. *Advances in Neural Information Processing Systems*.

Siheng Li, Zhanhui Zhou, Wai Lam, Chao Yang, and Chaochao Lu. 2025. Repo: Replay-enhanced policy optimization. *arXiv preprint arXiv:2506.09340*.

Zichuan Liu, Jinyu Wang, Lei Song, and Jiang Bian. 2025. Sample-efficient llm optimization with reset replay. *arXiv preprint arXiv:2508.06412*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*.

Leonard S Pleiss, Tobias Sutter, and Maximilian Schiffer. 2025. Reliability-adjusted prioritized experience replay. *arXiv preprint arXiv:2506.18482*.

Qwen Team. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Santhosh Kumar Ravindran. 2025. Cosmocore affective dream-replay reinforcement learning for code generation. *arXiv preprint arXiv:2510.18895*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*.

Yifan Sun, Jingyan Shen, Yibin Wang, Tianyu Chen, Zhendong Wang, Mingyuan Zhou, and Huan Zhang. 2025a. Improving data efficiency for llm reinforcement fine-tuning through difficulty-targeted online data selection and rollout replay. *arXiv preprint arXiv:2506.05316*.

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. 2025b. Delta: How does rl unlock and transfer new algorithms in llms? In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. 2020. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Harm Vanseijen and Rich Sutton. 2015. A deeper look at planning as learning from replay. In *International conference on machine learning*.

Chao Wang, Tao Yang, Hongtao Tian, Yunsheng Shi, Qiyao Ma, Xiaotao Liu, Ting Yao, and Wenbo Ding. 2025a. Learning more with less: A dynamic dual-level down-sampling framework for efficient policy optimization. *arXiv preprint arXiv:2509.22115*.

Chen Wang, Lai Wei, Yanzhi Zhang, Chenyang Shao, Zedong Dan, Weiran Huang, Yue Wang, and Yuzhi Zhang. 2025b. Eframe: Deeper reasoning via exploration-filtering-replay reinforcement learning framework. *arXiv preprint arXiv:2506.22200*.

Hong Wang, Zhezheng Hao, Jian Luo, Chenxing Wei, Yao Shu, Lei Liu, Qiang Lin, Hande Dong, and Jiawei Chen. 2025c. Scheduling your llm reinforcement learning with reasoning trees. *arXiv preprint arXiv:2510.24832*.

Junqiao Wang, Zeng Zhang, Yangfan He, Zihao Zhang, Xinyuan Song, Yuyang Song, Tianyu Shi, Yuchen Li, Hengyuan Xu, Kunyu Wu, and 1 others. 2024. Enhancing code llms with reinforcement learning in code generation: A survey. *arXiv preprint arXiv:2412.20367*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others.

2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Zed Industries. 2025. Zeta: Repository-level code edit prediction dataset. https://huggingface.co/datasets/zed-industries/zeta. Dataset, Hugging Face; Apache-2.0 license.

Hongzhi Zhang, Jia Fu, Jingyuan Zhang, Kai Fu, Qi Wang, Fuzheng Zhang, and Guorui Zhou. 2025a. Rlep: Reinforcement learning with experience replay for llm reasoning. *arXiv preprint arXiv:2507.07451*.

Jianqing Zhang, Zhezheng Hao, Wei Xia, Hande Dong, Hong Wang, Chenxing Wei, Yuyan Zhou, Yubin Qi, Qiang Lin, and Jian Cao. 2025b. Gapo: Robust advantage estimation for real-world code llms. *arXiv preprint arXiv:2510.21830*.

Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. 2023. Self-edit: Fault-aware code editor for code generation. *arXiv preprint arXiv:2305.04087*.

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. 2025. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? *arXiv preprint arXiv:2510.01161*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, and 1 others. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

A Additional Settings

We follow the default GRPO and DAPO settings in `verl`, with minor modifications for code-editing tasks: 4096-token input length, 1024-token output length, rollout batch size of 512, training batch size of 32, 8 rollouts per iteration, and 10 training epochs. For **A2R**, we set $K = 128$ (one-fourth of the rollout batch size) and $\lambda = 5$ by default. Because only a small portion of samples achieve an exact-match reward of 1, we set the RLEP reward threshold to 0.25, ensuring that over 80% of samples have at least two positive experiences available for replay. Otherwise, restricting replay to only successful experiences would leave no valid rollouts. All baselines use their recommended hyperparameters, and reported results are averaged over five independent runs.