CLEANER: SELF-PURIFIED TRAJECTORIES BOOST AGENTIC REINFORCEMENT LEARNING

Tianshi Xu
Peking University
tianshixu@stu.pku.edu.cn

Yuteng Chen
NTU, Singapore
yuteng003@e.ntu.edu.sg

Meng Li
Peking University
meng.li@pku.edu.cn

ABSTRACT

Agentic Reinforcement Learning (RL) has empowered Large Language Models (LLMs) to utilize tools like Python interpreters for complex problem-solving. However, for parameter-constrained models (e.g., 4B–7B), the exploration phase is often plagued by frequent execution failures, creating noisy trajectories that hinder policy optimization. Under standard outcome-based reward settings, this noise leads to a critical *credit assignment issue*, where erroneous actions are inadvertently reinforced alongside successful outcomes. Existing mitigations face a dilemma: dense rewards often trigger *reward hacking*, while supersampling incurs prohibitive computational costs. To address these challenges, we propose **CLEANER**. Distinct from external filtering methods, CLEANER exploits the model's intrinsic self-correction capabilities to eliminate error-contaminated context directly during data collection. At its core, the **Similarity-Aware Adaptive Rollback (SAAR)** mechanism autonomously constructs clean, purified trajectories by retrospectively replacing failures with successful self-corrections. Based on semantic similarity, SAAR adaptively regulates replacement granularity from shallow execution repairs to deep reasoning substitutions. By training on these self-purified paths, the model internalizes correct reasoning patterns rather than error-recovery loops. Empirical results on AIME24/25, GPQA, and LiveCodeBench show average accuracy gains of **6**%, **3**%, and **5**% over baselines. Notably, CLEANER matches state-of-the-art performance using only **one-third** of the training steps, highlighting trajectory purification as a scalable solution for efficient agentic RL. Our models and code are available at GitHub.

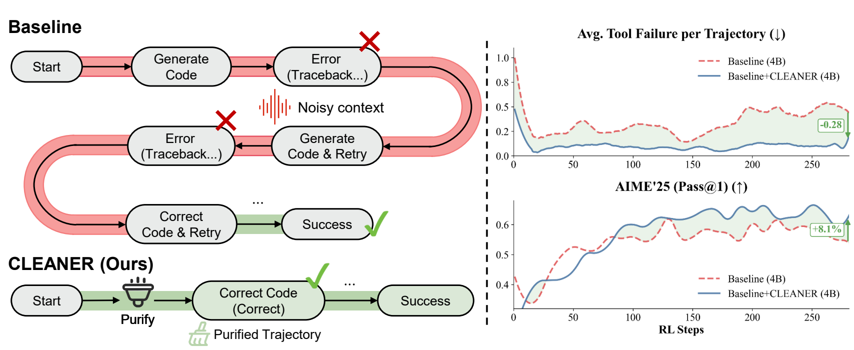

Figure 1: **Left**: Illustration of the differences between the standard baseline and our CLEANER. **Right**: By reducing the number of tool execution failures within trajectories during training, our method improves pass@1 accuracy on AIME'25 by 8.1%.

1 INTRODUCTION

The landscape of Large Language Models (LLMs) is shifting from passive text generation systems toward autonomous agents that solve complex tasks through tool use (Yao et al., 2022; Gou et al., 2023; Schick et al., 2023; Jin et al., 2025; Li et al., 2025c; Feng et al., 2025). Among the diverse tool modalities available to LLM agents, the Python code interpreter plays a particularly critical role Wang et al. (2024b); Shang et al. (2025); Yu et al. (2025b). Due to its Turing completeness and deterministic execution semantics, Python is indispensable for tasks that require precise computation, including mathematical reasoning, algorithmic problem-solving, and data analysis. As observed by Ronacher (Ronacher, 2025), code is increasingly serving as a "universal interface" that unifies logical reasoning, computation, and API interaction within a single expressive medium. This perspective aligns with frameworks like CodeAct (Wang et al., 2024b), which advocate for treating code execution as a first-class action in agentic reasoning. By doing so, agents can effectively plan, verify intermediate results, and iteratively correct errors through interaction with an execution environment. Together, these insights highlight that robust code synthesis and execution are foundational capabilities for tool-augmented LLM agents. Motivated by this perspective, this work focuses on *Python code execution* as the primary tool modality.

However, fully realizing this potential presents significant challenges for parameter-constrained models (e.g., 4B–7B). A primary obstacle is the high rate of execution failure, particularly during the exploration phase of Reinforcement Learning (RL) (Guo et al., 2025). Before policy convergence, these models frequently generate invalid code, causing the intended recovery mechanism to degenerate into prolonged "Error $\rightarrow$ Feedback $\rightarrow$ Retry" loops, as depicted in Figure 1 (left). This instability constitutes a critical bottleneck in training. As evidenced by the experimental results in Figure 1 (right), an excessive accumulation of tool errors within trajectories closely correlates with performance bottlenecks or even accuracy degradation. We attribute this to the pollution of context: repeated failures generate large amounts of misleading signals (e.g., invalid code and verbose tracebacks). This accumulated noise likely causes semantic interference, biasing the model toward rationalizing incorrect execution paths rather than re-grounding its decisions, thereby hindering policy improvement.

In principle, RL algorithms are expected to guide models away from such instability (Yu et al., 2025a; Chen et al., 2025). However, standard training paradigms frequently worsen the problem due to the **credit assignment issue**. Under sparse, outcome-based reward settings like GRPO Guo et al. (2025), the entire trajectory receives a uniform positive reward upon final success, regardless of preceding failures. This mechanism fails to distinguish between efficient reasoning and trajectories containing errors, effectively treating them as equivalent. Consequently, erroneous tool usage and the underlying logic are inadvertently reinforced despite their negative impact on reasoning.

To mitigate this, prior research has explored various strategies, yet each introduces new flaws. Attempts to assign dense rewards for individual tool executions often suffer from *reward hacking*, biasing agents toward optimizing intermediate metrics rather than final outcomes Yu et al. (2025b). Alternatively, works such as *rstar2-agent* (Shang et al., 2025) utilize supersampling-based trajectory filtering, retaining only high-quality instances from $2\times$ generated candidates. However, this incurs a prohibitive computational cost. Since the rollout phase usually dominates RL training (accounting for $> 80\%$ of runtime (Li et al., 2023; Sheng et al., 2024; Fu et al., 2025)), such extensive sampling renders these strategies unscalable for resource-constrained settings.

To address these challenges, we propose **CLEANER** (*Self-Purified Trajectories Boost Agentic Reinforcement Learning*). CLEANER significantly boosts the agentic RL by eliminating error-contaminated context from the training data. Unlike methods that rely on increasing rollout multiplicity, CLEANER operates specifically at the data level to refine the trajectories used for policy optimization. At the core of our approach is the **Similarity-Aware Adaptive Rollback (SAAR)** mechanism, which constructs self-purified trajectories. When the model generates incorrect code but subsequently self-corrects within the same rollout, SAAR intervenes to prevent the error-laden history from being used for optimization. Instead, it applies a retrospective context substitution where the trajectory is rolled back to the failure point and the erroneous action is replaced with the corrected solution. This process yields a revised trajectory containing substantially fewer execution errors. To ensure semantic coherence, SAAR adaptively regulates the rollback granularity based on the semantic similarity between the erroneous code and its corrected counterpart. High-similarity cases typically correspond to minor execution errors and trigger a shallow replacement that preserves the original reasoning. Conversely, low-similarity cases signal deeper logical flaws and necessitate the substitution of the entire reasoning segment to maintain consistency. By leveraging these self-purified trajectories, CLEANER reduces noise in the learning signal and accelerates capability acquisition. Empirical evaluations show that CLEANER outperforms standard baselines with average accuracy gains of approximately 6% on AIME, 3% on GPQA Rein et al. (2024), and 5% on LiveCodeBench Jain et al. (2024). Furthermore, it matches the performance of state-of-the-art (SOTA) models Yu et al. (2025b) while requiring only one-third of the RL steps. In summary, our main contributions are as follows:

❶ We propose **CLEANER**, which resolves the credit assignment dilemma in agentic RL by training on *self-purified trajectories*. This approach enables models to directly internalize correct reasoning patterns while filtering out the interference of execution noise.

❷ We introduce the **SAAR** mechanism to autonomously construct these clean signals. SAAR adaptively repairs failures—ranging from minor syntax typos to deep logical flaws—without the computational overhead of supersampling.

❸ We demonstrate that CLEANER achieves state-of-the-art efficiency and performance. It outperforms baselines with accuracy gains of 6% on AIME and 5% on LiveCodeBench, and notably matches SOTA performance using only **one-third** of the training steps.

❹ We provide a fully reproducible training pipeline and have made our code, environment configurations, and processed datasets available via Anonymous GitHub to support further research.

2 Preliminaries

2.1 Agentic Reasoning Trajectories

Notation. We formalize the agent's problem-solving process as a sequential generation task over a growing trajectory history. Let $\mathcal{M}$ denote the large language model acting as the agent, and let $\mathcal{E}$ denote the code execution environment (i.e., a Python interpreter). At turn t, the interaction history is denoted by h_t, which consists of the initial user query x and a sequence of past interaction tuples:

$$h_t = \big[x,\, (r_0, c_0, o_0), \ldots, (r_{t-1}, c_{t-1}, o_{t-1})\big]. \tag{1}$$

For each turn i, r_i denotes the *reasoning trace* expressed in natural language, c_i denotes the *code action* corresponding to an executable Python program, and o_i denotes the *observation* returned by the execution environment, i.e., $o_i = \mathcal{E}(c_i)$. We distinguish between successful executions, denoted by o_i^+, and execution failures or runtime errors, denoted by o_i^-.

Standard Generation Process. At step t, the policy π_θ conditions on the current history h_t and generates a reasoning trace and a code action:

$$(r_t, c_t) \sim \pi_\theta(\cdot \mid h_t). \tag{2}$$

The environment then executes the generated code and returns an observation $o_t = \mathcal{E}(c_t)$. The interaction history is updated by appending the new tuple:

$$h_{t+1} = h_t \oplus (r_t, c_t, o_t), \tag{3}$$

where $\oplus$ denotes sequence concatenation. In standard training pipelines, execution failures (o_t^-) are permanently recorded in the history, thereby introducing error-induced noise into subsequent conditioning and the resulting learning signal.

2.2 Group-Based Policy Optimization with Outcome-only Reward

We employ *sparse, outcome-only supervision*, rewarding agents solely upon trajectory completion. This avoids manual reward engineering and prevents "reward hacking" (Shang et al., 2025). For policy optimization, we use Group Relative Policy Optimization (Guo et al., 2025) to manage credit assignment via group-relative comparisons. See Appendix A for mathematical formulations.

3 Problem Formulation: Impact of Code Tool Execution Noise

Unlike internal Chain-of-Thought reasoning, agentic workflows introduce external stochasticity via interactions with the environment $\mathcal{E}$. This uncertainty manifests as trajectory-level noise that hinders efficient policy optimization.

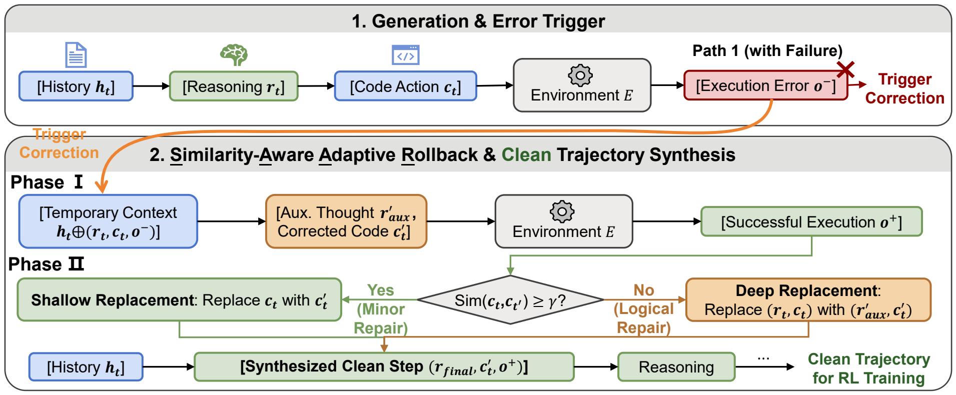

Figure 3: Illustration of our Similarity-Aware Adaptive Rollback (SAAR).

Context Contamination from Erroneous Tool Calls. Code execution is inherently error-prone, and failed executions ($o_t = o^-$) frequently occur during exploration. While these error traces contribute minimally to the final task resolution, they are permanently appended to the trajectory history h_t. Consequently, these low-information segments consume valuable context window capacity and disrupt the logical flow of subsequent reasoning, effectively contaminating the agent's decision context with noise.

Credit Assignment Ambiguity under Outcome-Only Reward. This uncertainty becomes particularly detrimental under sparse, outcome-based RL, where rewards are assigned solely based on final task success. As illustrated in Figure 2, we observe a distinct phenomenon: *bursts of erroneous tool calls within individual trajectories*. During these periods, accuracy plateaus or even degrades, indicating an optimization bottleneck. The root cause of this inefficiency lies in a fundamental **credit assignment failure** within *noisy successes*—trajectories that eventually succeed despite containing intermediate errors. Consider a typical noisy trajectory τ_{noisy}, in which the agent initially produces incorrect code but later self-corrects:

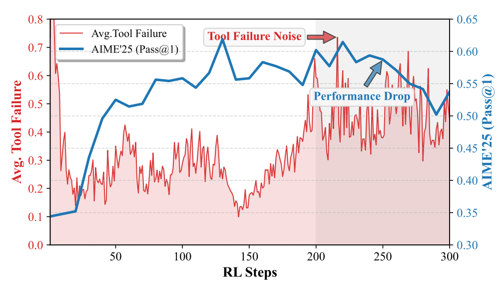

Figure 2: Impact of execution noise. Tool failure spikes correlate with AIME25 performance drops, highlighting optimization sensitivity to noisy trajectories.

$$\tau_{\text{noisy}} = [\ldots, h_t, \underbrace{(r_t, c_{\text{err}}, o^-)}_{\text{Noise (Trial 1)}}, \underbrace{(r'_{\text{aux}}, c_{\text{corr}}, o^+)}_{\text{Signal (Trial 2)}}, \ldots] \tag{4}$$

Here, the agent first emits an erroneous code action c_{err}, receives a runtime error o^-, and subsequently generates a corrected code c_{corr} accompanied by an auxiliary reasoning trace r'_{aux}, which executes successfully. Since the reward function $R(\tau)$ is binary and episodic, the final positive reward is uniformly propagated across the entire trajectory τ_{noisy}. Consequently, both the erroneous action c_{err} and the corrective action c_{corr} receive identical positive reinforcement, despite their fundamentally conflicting semantic roles. We refer to this effect as **Trajectory Noise**: spurious credit assigned to intermediate failures that dilutes the learning signal. Over time, this noise implicitly validates suboptimal tool usage patterns, amplifies variance in policy updates, and leads to brittle optimization.

4 METHOD: SIMILARITY-AWARE ADAPTIVE ROLLBACK (SAAR)

To boost agentic reinforcement learning under noisy tool interactions, we propose CLEANER, a trajectory purification framework centered on *Similarity-Aware Adaptive Rollback (SAAR)*. The core objective is to distill the learning signal by retrospectively eliminating execution failures from exploration rollouts, thereby constructing *clean, self-purified trajectories*. In these synthesized paths, the agent appears to solve the task fluently, enabling the optimizer to reinforce correct reasoning logic rather than error-recovery loops. As illustrated in Figure 3, this data-level intervention is triggered by execution errors and operates through a two-phase process:

4.1 PHASE I: ERROR TRIGGER AND LOOKAHEAD CORRECTION

At time step t, when the environment returns an execution error o_t^- following code action c_t, we defer committing this failure to the history. Instead, we freeze the current state h_t and initiate a temporary lookahead phase to seek a viable solution.

Context Extension. We temporarily construct an augmented context that exposes the execution error, allowing the model to analyze the feedback:

$$\tilde{h}_t = h_t \oplus (r_t, c_t, o_t^-). \tag{5}$$

Correction Generation. Conditioned on $\tilde{h}_t$, the policy generates a corrective response, typically comprising an auxiliary reasoning trace r'_{aux} and a revised code action c'_t:

$$r'_{\text{aux}}, c'_t \sim \pi_\theta(\cdot \mid \tilde{h}_t). \tag{6}$$

Verification. The revised code is executed to obtain a new observation $o'_t = \mathcal{E}(c'_t)$. If execution succeeds ($o'_t = o^+$), we proceed to Phase II to integrate this success into the trajectory. If failure persists, the correction loop repeats up to K attempts.

4.2 PHASE II: SIMILARITY-AWARE ADAPTIVE REPLACEMENT

Upon obtaining a valid correction c'_t, SAAR determines the optimal strategy to merge it into the history h_t. The intuition is that the semantic distance between the error c_t and correction c'_t reveals the nature of the failure. We quantify this using a similarity function $\mathrm{Sim}(c_t, c'_t)$, implemented via $\mathrm{difflib.SequenceMatcher}$, and compare it against a code similarity threshold γ.

Case A: Implementation-Level Repair ($\mathrm{Sim}(c_t, c'_t) \geq \gamma$). High similarity indicates a superficial error (e.g., syntax typos), where the original reasoning r_t is presumed to be sound. In this scenario, we perform a shallow replacement: the failed action c_t and error o_t^- are discarded, and the corrected code c'_t is grafted directly onto the existing reasoning r_t. This yields the purified tuple $(r_t, \mathbf{c'_t}, o'_t)$.

Case B: Reasoning-Level Repair ($\mathrm{Sim}(c_t, c'_t) < \gamma$). Low similarity signals a substantial divergence in implementation strategy, suggesting that the initial reasoning r_t is likely incompatible or misaligned with the corrected solution. Retaining the outdated reasoning would introduce semantic dissonance within the training data. Thus, we execute a *deep replacement*: the entire failed turn (r_t, c_t, o_t^-) is removed, and the auxiliary correction thought r'_{aux} is adopted as the canonical reasoning, forming the consistent tuple $(\mathbf{r'_{\text{aux}}}, \mathbf{c'_t}, o'_t)$.

Through this adaptive mechanism, we synthesize the *self-purified trajectory*:

$$\tau_{\text{purified}} = [\ldots, h_t, \underbrace{(\mathbf{r}_{\text{final}}, \mathbf{c'_t}, o'_t)}_{\text{Purified Context}}, \ldots], \tag{7}$$

where $r_{\text{final}} \in \{r_t, r'_{\text{aux}}\}$ is determined by the rollback granularity. This constructs a coherent, counterfactual history of immediate success, effectively guiding the policy to internalize correct reasoning patterns while bypassing the noise of trial-and-error.

4.3 IMPLEMENTATION DETAILS

Logit Recomputation via RadixAttention. Since the corrected action c'_t is sampled from the error-augmented context $\tilde{h}_t$, there exists a distribution shift: $\pi_\theta(c'_t \mid \tilde{h}_t) \neq \pi_\theta(c'_t \mid h_t \oplus r_{final})$. To ensure the policy update is grounded in the correct causal path, we must recompute the log-probabilities of c'_t under the purified context. To minimize overhead, we employ SGLang (Zheng et al., 2024) with *RadixAttention*. This mechanism efficiently reuses the KV cache for the invariant history prefix, restricting the computational cost strictly to the modified suffix segments.

Curriculum Mixing for Robustness. To balance error avoidance with error recovery, we employ a stochastic mixing strategy for Qwen2.5-7B. Specifically, we apply SAAR to 70% of trajectories while retaining 30% in their original state. This curriculum ensures the model internalizes correct reasoning patterns without sacrificing its intrinsic ability to debug and self-correct during inference.

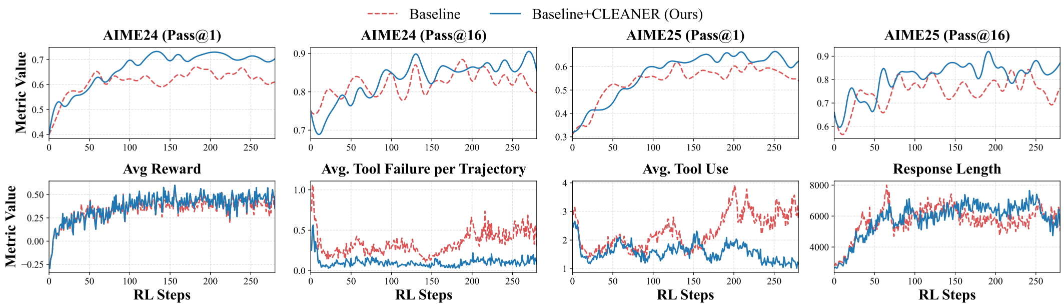

Figure 4: **Evolution of training metrics during RL.** Compared to the DAPO-baseline, CLEANER effectively suppresses erroneous tool calls in trajectories, leading to significant performance gains.

Table 1: **Comparing CLEANER with existing works. Bolded** entries denote the top-performing methods initialized from the same Qwen3-4B-Instruct base. Despite its compact 4B scale, CLEANER matches the performance of significantly larger models and achieves results comparable to SOTA baselines while requiring only **one-third** of the training steps utilized by DemyAgent-4B.

Method	**MATH**		**Science**	**LiveCodeBench**		**RL Step**
	AIME24	AIME25	GPQA	V6	Whole	(Batch Size=128)
Self-Contained Reasoning						
Qwen2.5-7B-Instruct	16.7	10.0	31.3	15.2	-	/
Qwen3-4B-Instruct-2507	63.3	47.4	52.0	35.1	-	/
Qwen2.5-72B-Instruct	18.9	15.0	49.0	-	-	/
DeepSeek-V3	39.2	28.8	59.1	16.1	49.6	/
DeepSeek-R1-Distill-32B	70.0	46.7	46.7	-	-	/
DeepSeek-R1-Zero (671B)	71.0	53.5	53.5	-	-	/
Agentic Reasoning						
ToRL-7B	43.3	30.0	-	-	-	550
ReTool-32B	72.5	54.3	-	-	-	1200
Tool-Star-3B	20.0	16.7	-	-	-	120
ARPO-7B	30.0	30.0	53.0	12.1	15.8	157
AEPO-7B	33.0	30.0	55.6	14.3	17.8	157
rStar2-Agent-14B	80.6	69.8	60.9	-	-	500
DemyAgent-4B (Qwen3-4B-Instruct)	72.6	**70.0**	58.5	26.8	51.7	750
DAPO-baseline (Qwen3-4B-Instruct)	66.7	59.4	56.9	26.6	49.5	250
CLEANER-4B (Qwen3-4B-Instruct)	**72.7**	67.1	**60.2**	**26.8**	**54.9**	**250**

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models and Training Datasets. We evaluate our framework using Qwen3-4B-Instruct-2507 (Yang et al., 2025) and Qwen2.5-7B-Instruct (Yang et al., 2024). The models first undergo cold-start Supervised Fine-Tuning (SFT) followed by reinforcement learning (RL), both utilizing the agentic datasets provided by (Yu et al., 2025b). Comprehensive dataset specifications and processing details are documented in Appendix B.

Implementation. We implement our training pipeline using the VeRL framework (Sheng et al., 2024) distributed via PyTorch FSDP2. We employ the code judge from (Shang et al., 2025) as the Python interpreter, which ensures robust stability even under the heavy concurrency of tool invocations during the RL rollout phase. Additionally, our prompt design adheres to the specifications outlined in (Yu et al., 2025b). Trajectory rollouts are generated using SGLang (Zheng et al., 2024). To address severe training instability caused by numerical inconsistencies between the training and rollout phases, we adopt FP16 precision for rollout generation following (Qi et al., 2025).

Training Recipe. We train the models for one epoch using the DAPO algorithm Yu et al. (2025a). We employ a rollout batch size of 128, a group size of 16, and an update mini-batch size of 32. The learning rate is set to 2e-6 for the 4B model and 1e-6 for the 7B model. Specifically for the Qwen2 experiments, we generate 8 rollouts per query and filter out instances that are either trivially easy or unsolvable to ensure stability. Further experimental details are provided in Appendix B.

Evaluation Benchmarks. To comprehensively demonstrate the improvements in reasoning and coding capabilities achieved by our method, we conduct evaluations across four challenging benchmarks: AIME24, AIME25, LiveCodeBench (Jain et al., 2024), and GPQA (Rein et al., 2024). To ensure a fair comparison, all models are evaluated using identical sampling parameters; detailed specifications are provided in the Appendix B.

Baselines. To rigorously assess the efficacy of CLEANER, we compare it against baselines across two distinct categories: **1)** *Self-Contained Reasoning.* We include standard instruction-tuned and reasoning-specialized models: Qwen2.5-7B-Instruct (Hui et al., 2024), Qwen3-4B-Instruct-2507 (Yang et al., 2025), Qwen2.5-72B-Instruct, DeepSeek-V3 (Liu et al., 2024), DeepSeek-R1-Distill-32B, and DeepSeek-R1-Zero (671B) (Guo et al., 2025). **2)** *Agentic Reasoning.* We compare against SOTA agentic models, including ToRL-7B (Li et al., 2025c), ReTool-32B (Feng et al., 2025), Tool-Star-3B (Dong et al., 2025b), ARPO-7B (Dong et al., 2025c), AEPO-7B (Dong et al., 2025a), Demystify-4B (Yu et al., 2025b), and rStar2-Agent-14B (Shang et al., 2025). We also include a DAPO-baseline that shares an identical training configuration with CLEANER, with the sole exception of excluding the SAAR mechanism.

5.2 Main Results

CLEANER Converts Execution Noise into Effective Reasoning. Figure 4 illustrates the evolution of key metrics for both the DAPO-baseline and CLEANER during RL. Empirical results support three primary conclusions: **1) Error Suppression:** Through SAAR, CLEANER consistently suppresses erroneous tool calls to a minimal level, mitigating interference with the model's reasoning process. **2) Performance Gains:** Reduced noise translates to significant improvements on AIME24/25 (avg. +6% Pass@1, +8% Pass@16), demonstrating enhanced exploration. **3) Efficient Reasoning:** Despite comparable output lengths, the reduction in errors implies that CLEANER reallocates tokens from futile tool calls to effective reasoning, facilitating deeper thinking.

Compared to Previous Works. The main results comparing CLEANER with existing works are summarized in Table 1. We observe the following: **1)** Compared to Self-Contained Reasoning models that lack specialized training for agentic scenarios, CLEANER demonstrates robust performance despite its compact 4B parameter size. This validates that small models, when subject to tailored post-training, can achieve capabilities comparable to significantly larger counterparts. **2)** In contrast to the SOTA baseline DemyAgent-4B, CLEANER attains comparable results using only **one-third** of the training steps. Notably, it surpasses it on AIME24, GPQA, and LiveCodeBench. We attribute this efficiency to the purified trajectories, which enable the model to acquire coding and reasoning capabilities more rapidly and effectively. Conversely, the DAPO-baseline exhibits significantly lower accuracy under limited training (250 steps), due to interference from tool call noise.

5.3 Ablation Study

Ablation on the Effectiveness of CLEANER. As detailed in Table 2, we evaluate three configurations under identical hyperparameters to isolate the contribution of each component: (1) **RL w/o Tools**, relying solely on internal reasoning; (2) **RL w/ Tools**, which integrates a Python code interpreter; and (3) **RL w/ Tools + SAAR** (i.e., CLEANER). The results yield two key observations: **1) The necessity of tool integration.** Equipping the model with a code interpreter significantly enhances performance on mathematical and coding tasks, improving average accuracy by over 5% on Qwen3-4B and 20% on Qwen2.5-7B. **2) The superiority of purified trajectories.** CLEANER consistently outperforms the baselines across all benchmarks and model scales. Specifically, for Qwen3-4B, we achieve average gains of 6% on AIME, 4% on GPQA, and 5% on LiveCodeBench. Similarly, Qwen2.5-7B exhibits an average improvement of 4%. These findings confirm that the trajectory purification effectively amplifies the potential of Agentic RL.

Table 2: Ablation study on the effectiveness of CLEANER.

Method	AIME24 Pass@1/16	AIME25 Pass@1/16	GPQA	LiveCodeBench-v6
Qwen3-4B-Instruct				
RL w/o Tools	64.0/78.8	53.3/77.0	52.2	19.8
+ Tools	66.7/84.4	59.4/84.2	56.9	26.6
+ SAAR (Ours)	**72.7/87.6**	**67.1**/84.1	**60.2**	**26.8**
Qwen2.5-7B-Instruct				
RL w/o Tool	15.4/30.5	14.4/24.4	32.3	1.1
+ Tools	40.2/59.1	27.3/46.3	35.9	13.0
+ SAAR (Ours)	**44.6/64.3**	**31.0/54.7**	**40.0**	**13.1**

Table 3: Ablation on learning rate.

Method	AIME24 Pass@1/16	AIME25 Pass@1/16
RL w/ Tools (1e-6)	66.9/85.1	63.3/83.9
CLEANER-4B (1e-6)	70.8/85.4	64.2/86.4
CLEANER-4B (2e-6)	72.7/87.6	67.1/84.1

Table 4: Ablation on SAAR deactivation. Performance and the evaluation time comparison with vs. without SAAR during the evaluation phase.

Method	AIME24 Pass@1/16	AIME25 Pass@1/16	GPQA	LiveCodeBench-v6	Time (min)
CLEANER-4B	72.7/87.6	67.1/84.1	60.2	26.8	115
CLEANER-4B w/o SAAR	72.1/86.3	64.6/84.3	59.8	26.6	106

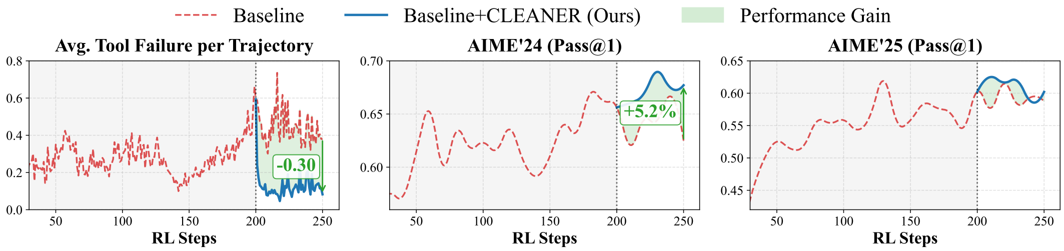

Figure 5: **Recovery from suboptimal policies.** Comparison of training metrics before and after introducing CLEANER at step 200. The inclusion of CLEANER effectively stabilizes the optimization process, leading to a marked improvement in final performance.

Ablation on Learning Rate. Table 3 summarizes our ablation study on learning rates across different model scales. For the 4B model, we adopted a relatively large learning rate of 2e-6 to accelerate convergence. As summarized in Table 3, CLEANER achieves consistent improvements across different settings, with 2e-6 yielding superior results. For the 7B model, we adopted a learning rate of 1e-6 to ensure optimization stability for the larger parameter space.

Internalization vs. Scaffolding. To verify that CLEANER effectively internalizes reasoning patterns, we evaluate the model with the SAAR mechanism deactivated. As detailed in Table 4, the model retains robust performance even in the absence of this "scaffolding." Specifically, on AIME24, Pass@1 and Pass@16 decline by only 0.6% and 1.3%, respectively; on AIME25, Pass@1 decreases by 2.5%, while Pass@16 exhibits a marginal gain of 0.2%. Similarly, GPQA and LiveCodeBench show negligible performance degradation. This confirms that the policy has assimilated the error-avoidance logic into its intrinsic parameters, enabling efficient deployment without external dependencies. Alternatively, SAAR can serve as a lightweight inference enhancement. It incurs a mere 8.8% increase in average latency—significantly lower than computational-heavy test-time scaling methods like tree search (Bi et al., 2024; Yao et al., 2023)—while effectively mitigating in-context code errors and improving stability.

Recovery from Suboptimal Policies. To assess the restorative capability of our method, we performed a recovery experiment initializing from step 200 of the DAPO-baseline. At this time, the baseline exhibited significant instability, averaging 0.6 erroneous tool invocations per trajectory. As shown in Fig. 5, CLEANER quickly stabilized training, effectively suppressing erroneous invocations and reducing the average number of tool calls per trajectory. Consequently, accuracy on AIME24 and AIME25 improved by 5.2% and 1.0%, respectively. However, we observe that this post-hoc recovery failed to reach parity with models trained with SAAR from scratch, underscoring the necessity of integrating the mechanism throughout the entire training lifecycle.

6 Conclusion

To address the inefficiency in agentic RL caused by execution noise and ambiguous credit assignment. We propose CLEANER, which employs Similarity-Aware Adaptive Rollback to transform noisy exploration logs into clean, self-purified trajectories prior to optimization. This aligns training signals with correct behavior, enabling models to internalize robust tool usage without error interference. Empirical results on AIME24/25, GPQA, and LiveCodeBench show average accuracy gains of 6%, 3%, and 5% over baselines. Crucially, CLEANER matches the performance of SOTA methods while requiring only one-third of the training steps, highlighting trajectory purification as a scalable alternative for efficient agentic RL.

REFERENCES

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, et al. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*, 2025a.

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025b.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025c.

R Z Fan, Z Wang, and P Liu. Megascience: Pushing the frontiers of post-training datasets for science reasoning. *arXiv preprint arXiv:2507.16812*, 2025.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jujie He et al. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 2025.

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

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, et al. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025a.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*, 2025b.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025c.

Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=hO0c2jD5c3`.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via fp16. *arXiv preprint arXiv:2510.26788*, 2025.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Armin Ronacher. Building an agent that leverages throwaway code, 2025. URL `https://lucumr.pocoo.org/2025/10/17/code/`.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Ning Shang, Yifei Liu, Yi Zhu, Li Lyna Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng Dong, Xudong Zhou, Bowen Zhang, et al. rstar2-agent: Agentic reasoning technical report. *arXiv preprint arXiv:2508.20722*, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. *arXiv preprint arXiv:2505.01441*, 2025.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*, 2025.

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowledge comes from practice: Aligning llms with embodied environments via reinforcement learning. *arXiv preprint arXiv:2401.14151*, 2024.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Otc: Optimal tool calls via reinforcement learning. *arXiv e-prints*, pp. arXiv–2504, 2025a.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. *arXiv preprint arXiv:2410.14803*, 2024a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024b.

Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024c.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

An Yang, Baosong Yang, Binyuan Hui, Bo He, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Deng, Dayiheng Liu, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

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

Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan, and Mengdi Wang. Demystifying reinforcement learning in agentic reasoning. *arXiv preprint arXiv:2510.11701*, 2025b.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

A Detail Preliminaries

To optimize the policy efficiently without the overhead of a value function critic, we operate within the **Group Relative Policy Optimization (GRPO) framework** Guo et al. (2025). This paradigm estimates the baseline from group statistics rather than a separate neural network. Specifically, for each query q, a group of G trajectories $\{\tau_i\}_{i=1}^G$ is sampled from the current policy $\pi_{\theta_{old}}$. The advantage A_i for the i-th trajectory is derived by normalizing its reward against the group statistics:

$$A_i = \frac{R(\tau_i) - \mu_R}{\sigma_R + \delta}, \tag{8}$$

where μ_R and σ_R denote the mean and standard deviation of the group rewards, respectively. Following this formulation, the policy is updated by maximizing the surrogate objective:

$$\begin{aligned}\mathcal{J}(\theta) = \mathbb{E}_{\substack{q\sim P(Q)\\ \tau\sim\pi_{\theta_{old}}}} \Bigg[\frac{1}{G}\sum_{i=1}^{G} \min \Big(\rho_i A_i, \\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon) A_i \Big) \Bigg],\end{aligned} \tag{9}$$

where $\rho_i = \frac{\pi_\theta(\tau_i|q)}{\pi_{\theta_{old}}(\tau_i|q)}$ represents the importance sampling ratio, and ϵ is the clipping hyperparameter. This objective serves as the optimization backbone for our training process.

B Implementation Details

Table 5: Hyperparameters for Reinforcement Learning.

Hyperparameter	Value
Learning Rate	2×10^{-6} (4B) / 1×10^{-6} (7B)
Max Prompt Length	2,560
Max Response Length	20,480 (Avg. $\approx$ 7,000)
LR Warmup Steps	20
PPO Clip Ratio (ϵ^-, ϵ^+)	$0.20, 0.28$
Retry Limit K	3
Similarity Threshold γ	0.5
Reward Type	Outcome-only $\{-1, 1\}$

Table 6: Sampling configurations for evaluation.

Hyperparameter	Value
Temperature	1.0
Top-p	0.6
Top-k	-1

Models and Training Datasets For the RL phase, we utilize the open-source dataset from (Yu et al., 2025b) which comprises a diverse mixture of 17k samples from DAPO-Math (Yu et al., 2025a), 4,902 math and 3,586 code samples from Skywork-or1 (He et al., 2025), and 3k science problems from MegaScience (Fan et al., 2025).

Training Configurations Table 5 summarizes the hyperparameters for our reinforcement learning stage. We employ different learning rates for models of varying scales: 2×10^{-6} for the 4B model and 1×10^{-6} for the 7B model. Although the maximum context window is set to 20,480 tokens to accommodate long-horizon trajectories, the empirical average sequence length across our training data remains approximately 7,000 tokens, ensuring computational efficiency without sacrificing context.

Ablation on Key Parameters The retry limit K and similarity threshold γ are critical for the CLEANER framework. Through empirical validation, we found that $K = 3$ offers an optimal balance between recovery rate and computational cost; values below 3 lead to a noticeable drop in the recovery of successful trajectories, while values above 3 yield diminishing returns. Regarding the similarity threshold γ, our method exhibits strong robustness across a range of values. However, we recommend a relatively high threshold (between 0.5 and 1.0) to ensure high-fidelity trajectory refinement, with 0.5 being our default setting.

Hardware and Compute Costs All experiments were conducted on a single node equipped with 4×NVIDIA H100 or 4×NVIDIA H200 GPUs. For the Qwen3-4B model, the full training cycle takes approximately 4 days. Interestingly, training the Qwen2.5-7B model requires only 2 days. This is primarily due to our data filter process, where we filter out both overly simplistic and excessively difficult samples to focus the training on more informative trajectories.

Evaluation Sampling Parameters Table 6 lists the specific decoding configurations used during the evaluation phase to ensure consistent performance comparison.

C Unsuccessful Attempts: Leveraging Negative Samples via SAAR

During the development of the **CLEANER** framework, we explored an alternative strategy to further enhance model performance by utilizing the erroneous actions identified by the **SAAR** mechanism as negative samples. Despite investigating multiple configurations, these attempts did not yield the expected improvements. We share these findings here to provide insights for future research in agentic RL.

Motivation The **SAAR** mechanism primarily functions by overwriting failed tool invocations with correct actions, ensuring that the agent is exposed to high-quality, "purified" trajectories during training. We hypothesized that the original erroneous actions, which are discarded in the standard CLEANER pipeline, could serve as valuable negative signals. By explicitly learning what constitutes an incorrect behavior through contrastive signals, the model might further refine its coding and tool-use capabilities.

Implementation For each successful trajectory recovery via **SAAR**, we paired the original failed tool call with the corrected rollout to generate online positive-negative pairs. We then applied an **online-DPO** (Direct Preference Optimization) objective to these pairs. To isolate the impact of the tool call itself, we masked all tokens except for the tool invocation segment, penalizing the failed attempt while rewarding the corrected one. During optimization, the DPO gradient was integrated with the GRPO signal at each step to perform a unified policy update.

Results and Analysis While this approach slightly improved the model's ability to "self-repair" specific code snippets, it failed to improve the overall success rate on complex reasoning tasks and even triggered training collapse in the later stages. Our analysis suggests several reasons for this failure:

❶ **Reasoning vs. Syntactic Failure:** Tool invocation failures in advanced agents are frequently rooted in faulty reasoning rather than mere syntax errors. As training progresses, the **SAAR** mechanism shifts from addressing minor typos toward "deep" logical repairs. Penalizing only the tool call segment without addressing the preceding CoT creates a disconnect between the agent's internal logic and its external actions. This imbalance may discourage the generation of complex code instead of fostering better reasoning.

❷ **Correlation between Reasoning and Code Proficiency:** Code-use proficiency is intrinsically linked to the agent's overall reasoning capacity. We observed that as the model's reasoning improves, it naturally adopts more sophisticated code structures. Relying on simple token-level masking to reward/penalize specific segments can be counterproductive. Furthermore, recent studies (e.g., **GSPO** (Zheng et al., 2025)) indicate that assigning disproportionate weights to specific tokens within a single trajectory can adversely affect policy optimization stability.

Ultimately, effectively utilizing negative samples and determining their net value in agentic RL remains an open question for future study.

D Related Work

Static and Supervised Tool-Integrated Reasoning. Tool-integrated reasoning (TIR) empowers LLMs to offload precise computations to external environments (Parisi et al., 2022; Schick et al., 2023; Wang et al., 2024b). Foundational paradigms like ReAct (Yao et al., 2022) and Program of Thoughts (Chen et al., 2022) established the viability of interleaving reasoning with execution. Scaling these capabilities, recent Supervised Fine-Tuning (SFT) approaches (Gou et al., 2023; Qin et al., 2023; Schick et al., 2023) and unified executable frameworks like CodeAct (Wang et al., 2024b) have achieved remarkable performance by mimicking expert trajectories. However, relying solely on behavioral cloning limits models to successful demonstrations. Consequently, these agents often mimic surface-level patterns without grasping the underlying causality, leaving them ill-equipped to handle the inherent noise of real-world tool interactions. (Wang et al., 2024c; Kumar et al., 2024).

Agentic RL. To bridge the gap left by supervised methods, Agentic RL treats tool invocation (e.g., executable code (Wang et al., 2024b)) as an explicit action space, optimizing adaptive strategies via outcome-driven rewards (Shridhar et al., 2020; Mialon et al., 2024). This paradigm enables agents to move beyond simple imitation toward discovering flexible solutions in open-ended tasks (Tan et al., 2024; Bai et al., 2024; Wang et al., 2024a). Recent advancements have systematically scaled these capabilities to autonomous search and query refinement (Jin et al., 2025; Song et al., 2025; Sun et al., 2025), long-horizon research tasks (Li et al., 2025b;a), and complex multi-tool coordination (Singh et al., 2025; Dong et al., 2025b; Qian et al., 2025; Wang et al., 2025a;b). These developments are further supported by studies on scaling laws (Li et al., 2025c) and the strategic logic of tool invocation (Feng et al., 2025). Crucially, recent research has begun to prioritize fundamental training stability and exploration efficiency. While *Demystifying RL* (Yu et al., 2025b) investigates fundamental training recipes and *rStar2-Agent* (Shang et al., 2025) mitigates execution noise via trajectory filtering, ARPO (Dong et al., 2025c) and AEPO (Dong et al., 2025a) specifically focus on enhancing exploration. They introduce entropy-regulated mechanisms to dynamically modulate rollouts, leveraging model uncertainty to improve performance in multi-turn interactions. Despite these improvements, existing methods still struggle to effectively decouple high-quality signals from the pervasive noise inherent in complex tool-use trajectories, which often leads to sub-optimal policy updates. To address this, our CLEANER framework introduces a robust mechanism to refine training data and stabilize the learning process.