Towards Effective Code-Integrated Reasoning

Fei Bai[1,4*], **Yingqian Min**[1,4*], **Beichen Zhang**[1,4*], **Zhipeng Chen**[1,4]
Wayne Xin Zhao[1,4†], **Lei Fang**[2], **Zheng Liu**[3], **Zhongyuan Wang**[3], **Hongteng Xu**[1,4]

[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]DataCanvas Alaya NeW
[3]BAAI
[4]Beijing Key Laboratory of Research on Large Models and Intelligent Governance

Abstract

In this paper, we investigate code-integrated reasoning (CIR), where models generate code when necessary and integrate feedback by executing it through a code interpreter. To acquire this capability, models must learn when and how to use external code tools effectively, which is supported by tool-augmented reinforcement learning (RL). Despite its benefits, tool-augmented RL can still suffer from potential instability in the learning dynamics. In light of this challenge, we present a systematic approach ETIR (Effective TIR) to improving the training effectiveness and stability of tool-augmented RL for code-integrated reasoning. Specifically, we develop enhanced training strategies that balance exploration and stability, progressively building tool-use capabilities while improving reasoning performance. Through extensive experiments on five mainstream mathematical reasoning benchmarks, our model demonstrates significant performance improvements over multiple competitive baselines. Furthermore, we conduct an in-depth analysis of the mechanism of code-integrated reasoning, revealing several key insights, such as the extension of model's capability boundaries and the simultaneous improvement of reasoning efficiency through code integration. These findings underscore the potential of code-integrated reasoning as a scalable paradigm for advancing robust and efficient language model reasoning.

Introduction

Recent advances in large reasoning models (DeepSeek-AI et al. 2025; Team et al. 2025; MiniMax et al. 2025) have demonstrated significant performance gains through increased test-time computation. Unlike conventional language models, these models do not produce answers directly. Instead, they engage in a deliberative thought process—searching, proposing, verifying, and evaluating potential solutions to arrive at the correct answer. However, even with more sophisticated reasoning procedures, their capabilities remain fundamentally constrained by the inherent limitations of large language models (Zhao et al. 2025). Since they still rely on the same underlying training and generation mechanisms, longstanding issues, such as imprecise numerical computation and restricted knowledge coverage, persist, ultimately diminishing models' effectiveness.

To overcome these inherent constraints, augmenting LLMs with external tools has shown great promise. For instance, executable code generation enhances mathematical reasoning (Chen et al. 2025; Gou et al. 2024), while search engine integration mitigates knowledge gaps (Sun et al. 2025b). Building on these insights, recent work explores tool integration by prompting (Li et al. 2025b; Chen et al. 2023) or supervised fine-tuning (Li et al. 2025a). However, these methods tend to be rigid, hindering the model's ability to utilize tools flexibly. In contrast, *tool-augmented RL* enables adaptive interaction through reinforcement learning (Song et al. 2025a; Li, Zou, and Liu 2025; Zhang et al. 2025a), leading to more effective tool use.

However, the training process of tool-augmented RL models remains unstable and prone to collapse (Wang et al. 2025b). We identify three key challenges: 1) poorly defined interaction boundaries introduce noise, especially when tool calls are mistimed or feedback is misused; 2) distributional shifts from tool feedback disrupt learning consistency by diverging from the model's native reasoning; and 3) fixed interaction budgets lead to response homogenization, causing mode collapse and reduced diversity. While prior work has partially identified these issues and proposed corresponding solutions (Mai et al. 2025; Song et al. 2025b), it remains insufficient to ensure stable training.

To address the challenges in tool-augmented RL, we propose **ETIR** (Effective TIR), a method built on the code-integrated reasoning task. It enhances training stability and model–environment interaction through three key strategies: precise match of interaction boundary, external tool feedback masking, and progressive increase of interaction budget. On five challenging mathematical benchmarks, including AIME2024 and AIME2025, ETIR achieves an average score of 52.7, outperforming multiple strong baselines.

In addition, we analyze how tool-augmented RL, especially code-integrated reasoning, improves the reasoning capabilities of language models, a direction that has been largely underexplored in prior work. Existing studies typically focus on improving performance, while paying little attention to analyzing why and how code-integrated reasoning contributes to such improvements. To bridge this

*These authors contributed equally.
†Corresponding Author

gap, we examine its underlying mechanisms in detail. By leveraging external interpreters, code integration extends the model's capacity boundary (measured as PASS@K) and enables more concise and structured solution paths. Notably, even when code is non-executable, it can still aid reasoning by encouraging the model to engage in revision. Moreover, the benefits of code integration vary across domains, revealing its task-specific effectiveness. These observations might provide useful insights for more in-depth future studies on tool-augmented reasoning.

To summarize, our contributions are three-fold:

• Our findings systematically reveal key factors that affect the stability of tool-augmented RL training, offering practical guidance for improving both the reliability and efficiency of training in such settings.

• We propose ETIR, a novel method that introduces progressive learning into tool-augmented RL for the first time, enabling more efficient model-environment interaction. Our approach outperforms multiple strong baselines on challenging mathematical benchmarks.

• We investigate the mechanisms underlying why and how code-integrated reasoning improves model performance, which are often neglected in prior work, providing empirical insights for future research.

Related Work

Tool-Integrated Reasoning. Tool-Integrated Reasoning allows large language models to leverage external tools to overcome their inherent limitations and enhance reasoning capacities. It mainly involves two types: 1) using code interpreters for mathematical reasoning (Gou et al. 2024) and 2) search engines for retrieval tasks (Li et al. 2025b). Common methods involve prompting-based tool invocation and supervised fine-tuning (SFT) with reasoning trajectories (Sun et al. 2025b; Li et al. 2025a; Lu et al. 2025; Chen et al. 2023; Goldie et al. 2025). While the latter is more effective than direct prompting in enhancing tool usage, it remains insufficient for achieving truly intelligent and adaptive interaction (Chen et al. 2025). Recently, advances in reinforcement learning have enabled models to acquire more flexible and adaptive tool usage (Mai et al. 2025; Feng et al. 2025; Li, Zou, and Liu 2025; Dong et al. 2025), though training stability remains a significant challenge. Beyond these advances, there is also a lack of systematic analysis on the underlying mechanisms through which tool integration enhances reasoning, which limits deeper understanding and the potential for further optimization of tool-augmented models.

Reinforcement Learning for LLMs. Reinforcement learning has emerged as a key method for aligning LLMs with human preferences, enabling models to go beyond conventional supervised learning. Previous approaches are dominated by PPO (Ouyang et al. 2022), which, while effective, is resource-intensive and sample-inefficient. To improve scalability, numerous improved reinforcement learning methods have emerged, effectively reducing training costs, enhancing sample efficiency, and introducing various optimization strategies (Shao et al. 2024; Yu et al. 2025; Hu et al. 2025; Yue et al. 2025b; Zhang et al. 2025b; Yuan et al. 2025; Cheng et al. 2025), thereby advancing the development of large reasoning models. However, studies have shown that vanilla reinforcement learning struggles to substantially expand the capability boundaries of models (Yue et al. 2025a; Wu et al. 2025), highlighting the promise of tool-augmented RL as a more effective alternative.

Preliminary

In this section, we introduce two key concepts: 1) code-integrated reasoning, where LLMs generate code and execute it through a code interpreter to solve complex tasks; 2) tool-augmented RL, which enhances LLMs by incorporating feedback from external tools to improve learning.

Code-Integrated Reasoning

We formalize *code-integrated reasoning*, where LLMs selectively invoke code execution during inference to handle tasks requiring precise numerical or symbolic computation.

Given a question Q, the model generates reasoning steps or code conditioned on Q and the current context H_t. Once executed by an external interpreter, the resulting outputs are reintegrated to guide subsequent reasoning. This iterative loop of code generation, execution, and context updates refines the model's reasoning and yields more accurate, verifiable outputs. The process is formally described as follows:

$$R_t \text{ or } C_t = f_\theta(Q, H_t), \tag{1}$$

$$O_t = \mathcal{I}(C_t), \tag{2}$$

$$H_{t+1} = H_t \oplus R_t \oplus C_t \oplus O_t, \tag{3}$$

$$y = f_\theta(Q, H_T), \tag{4}$$

where t and T denote the current and final code-generation steps for answering Q, while R, C, and O represent the reasoning step, generated code, and execution result. The process starts from H_0 and proceeds step by step until producing the final answer y.

Tool-Augmented RL Framework

To develop our code-integrated reasoning model, we begin with base models and employ typical tool-augmented reinforcement learning (RL) for optimization.

Vanilla RL algorithms like PPO optimize LLMs by maximizing the following surrogate objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{q\sim P(Q),\, o\sim\pi_{\theta_{\text{old}}}(O|q)}\left[\frac{1}{|o|}\sum_{t=1}^{|o|}\min\Big(s_t(\theta)A_t,\right.$$

$$\left.\text{clip}(s_t(\theta), 1-\epsilon, 1+\epsilon)A_t\Big)\right], \tag{5}$$

where $s_t(\theta)$ represents the probability ratio between the new policy and old policy for observation o_t:

$$s_t(\theta) = \frac{\pi_\theta(o_t|q, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t|q, o_{<t})}, \tag{6}$$

The trajectory $o = [o_0, o_1, ..., o_T]$ reflects only the model's autoregressively generated intrinsic knowledge, excluding any external feedback.

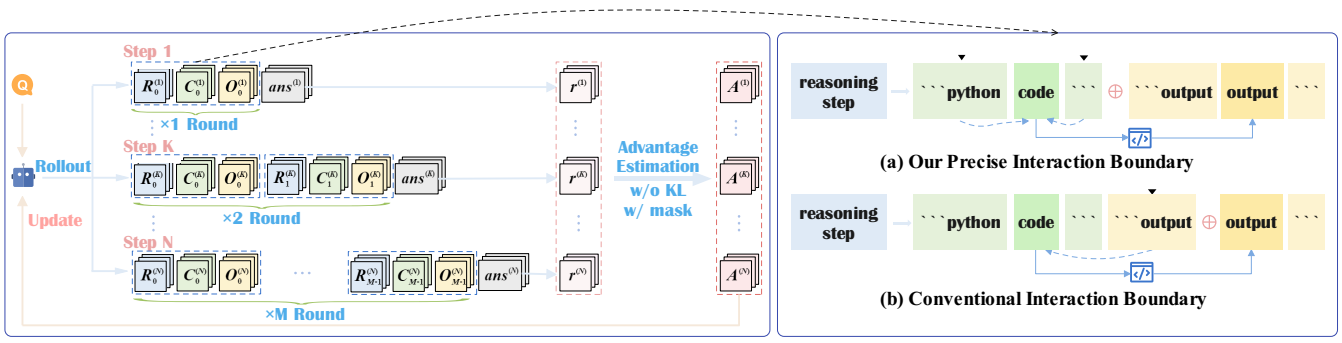


Figure 1: The overview of ETIR, illustrating the progressive increase in interaction budget and the precise match of interaction boundaries during training.

In contrast, the tool-augmented RL framework enables multi-round interaction with external tools during rollouts, generating trajectories enriched with interaction information. This modifies $s_t(\theta)$ as:

$$s_t'(\theta) = \frac{\pi_\theta(o_t|q, o_{<t}; \mathcal{I})}{\pi_{\theta_{\text{old}}}(o_t|q, o_{<t}; \mathcal{I})}, \tag{7}$$

where $\mathcal{I}$ represents the external tool, such as code interpreter. Through integrating feedback from external tools, the model can adjust its decision-making process based on concrete execution results, enhancing learning effectiveness.

Methodology

In this section, we first discuss the challenges encountered in the conventional tool-augmented reinforcement learning process, which hinder the model's performance from both stability and exploration perspectives, and then present our method, **ETIR**, which enhances both the effectiveness and stability of model interactions with external tools by employing a progressive learning approach. This method systematically refines the model's ability to engage with tools, ensuring more reliable and efficient performance over time.

Analysis of Challenges on Tool-augmented RL

Tool-augmented RL, due to its incorporation of model interactions with the external environment, is inherently more unstable compared to vanilla RL. During training, issues such as reward collapse, gradient norm explosion, and response length fluctuation frequently occur, signaling training failure and ultimately leading to a decline in model performance. We summarize the factors that impact training stability in tool-augmented RL from the following aspects.

Instability caused by interaction boundary disruptions. To integrate external tools effectively, clear interaction boundaries must be defined, which specify when to trigger tool execution and how to incorporate feedback. Improperly setting interaction boundaries can introduce noise. As shown in Figure 1(b), code execution is typically triggered when the model emits $\boxed{\text{“`}output}$, after which the result is appended and reasoning resumes. However, the model does not always generate $\boxed{\text{“`}output}$ after code, causing the reasoning process to terminate prematurely.

Impact of distributional shifts on learning. Distributional shifts arise between the model's own reasoning process and the feedback returned by external tools, disrupting the continuity of inference. The feedback might differ from the model's original reasoning patterns, thus interrupting the model's inference flow. As tool use becomes more frequent, these shifts accumulate over time, gradually undermining training stability and increasing the risk of collapse.

Response homogenization under fixed interaction budgets. In tool-augmented RL, the risk of response homogenization, commonly known as mode collapse, becomes more pronounced. Due to the fixed tool interaction rounds during training, the model tends to converge on a narrow range of behaviors that are consistently rewarded within the constrained interaction budget, which results in increasingly deterministic outputs, reducing the diversity of reasoning patterns explored by the model. As illustrated in Figure 2, if the interaction budget is too low, the model may fail to meet its tool-use needs during reasoning; if too high, it may produce redundant code to fill the quota. Both cases can lead to incorrect answers.

The Proposed Method: ETIR

To mitigate the instability in tool-augmented RL, we propose ETIR, an RL method that enables more efficient tool interaction learning in fewer steps. This method primarily consists of three components: 1) precise match of interaction boundary; 2) external tool feedback masking; 3) progressive increase of interaction budget.

Precise Match of Interaction Boundary. To mitigate the instability caused by improper tool interaction boundaries, we employ an exact match-based approach, setting a clearer interaction boundary to trigger the interaction with the environment (shown in Figure 1(a)). In detail, unlike conventional interaction boundary with single stop token $\boxed{\text{“`}output}$, which is prone to premature termination or inaccurate boundary detection, our approach ensures that tool interaction is only triggered when the system detects both

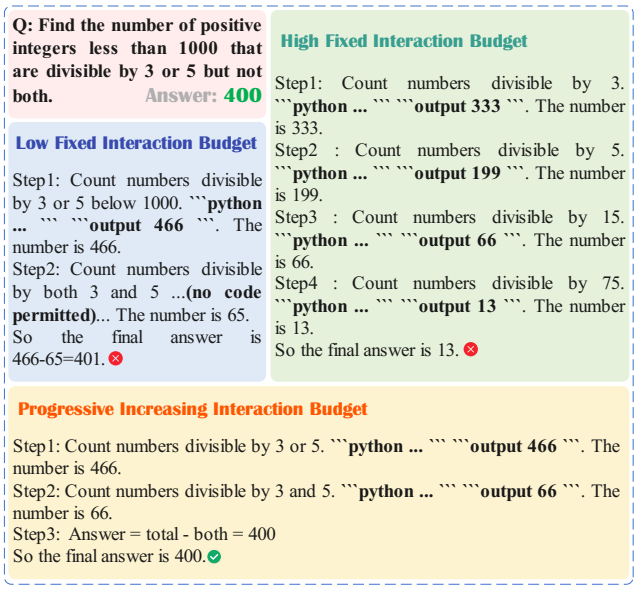


Figure 2: Comparison of different interaction budgets. A model trained with a progressively increasing interaction budget can adaptively interact with the environment during inference, making it more likely to produce correct answers.

“‘*python* and “‘ in the model's output meanwhile, after which the system extracts the code between these markers and passes it to the interpreter for execution. Consequently, this method reduces training noise by ensuring the model uses well-structured, consistently formatted code, improving feedback reliability and learning stability.

External Tool Feedback Masking. During model optimization, we mask the feedback from external tools when calculating the training loss to prevent the model from learning unstable and variable outputs. This reduces distribution mismatch between the model's own reasoning and external feedback, allowing the model to focus on critical reasoning patterns and enhancing training stability.

Progressive Increase of Interaction Budget. Fixed interaction budgets can lead the model to converge early during training. To address this, we gradually increase the number of tool invocations allowed throughout the training process, shown in Figure 1. This progressive scheduling strategy encourages the model to explore a broader range of reasoning paths in the early stages rather than settling too rapidly on narrowly optimized, high-reward behaviors. By delaying over-stabilization of the policy, the model maintains output diversity for a longer period, fostering more robust strategy learning. At training step N, the maximum interaction round is controlled by a scheduling function $B(N)$:

$$\begin{cases} B(N+1) \geq B(N), \\ B(0) = B_{\text{init}}, \\ B(N) \leq B_{\text{max}}, \end{cases} \tag{8}$$

When $B(N) = M$, the generated trajectory $\tau^{(N)}$ with the maximum interaction rounds is:

$$\tau^{(N)} = [R_0^{(N)}, C_0^{(N)}, O_0^{(N)}, \ldots, R_m^{(N)}, C_m^{(N)}, O_m^{(N)}, \ldots, R_{M-1}^{(N)}, C_{M-1}^{(N)}, O_{M-1}^{(N)}, ans], \tag{9}$$

where $R_m^{(N)}$ denotes the model's natural language reasoning step in the m-th interaction round, $C_m^{(N)}$ is the generated code for tool use, and $O_m^{(N)}$ is the execution result. The final prediction is denoted as ans. After completing the rollout, the model receives a reward $r^{(N)}$ based on the trajectory $\tau^{(N)}$ and computes the advantage $A^{(N)}$ to update the policy.

Experiments

In this section, we first detail the experimental setup and then report the results and detailed analysis.

Experiment Settings

Training. We employ Qwen2.5-Math-7B (denoted as QWEN2.5-M7) (Yang et al. 2024) as our backbone model. All training is conducted under the veRL framework (Sheng et al. 2025) using 38,000 samples from STILL-3 (Chen et al. 2025), which provides a balanced data difficulty distribution. We adopt the REINFORCE++ algorithm (Hu et al. 2025) with progressively increasing tool interaction rounds (2 to 4), removing the KL term and enabling entropy bonus. A binary reward of 1/-1 is given based on answer correctness.

Evaluation. To evaluate our method, we conduct experiments on five benchmarks: MATH500 (Hendrycks et al. 2021), AMC23, AIME2024, AIME2025, and OlymMATH (Sun et al. 2025a). These cover a range of difficulty levels from school math to Olympiad-level reasoning. For evaluation, we report AVG@16 (temperature=1.0) on AMC23, AIME2024 and 2025, and OlymMATH, and use greedy decoding for MATH500 due to its larger size.

Baselines. We compare our model with two types of baselines: 1) text-only models without code usage, including QWEN2.5-MATH-7B-INSTRUCT, a strong supervised model, and RL-optimized models such as SIMPLERL-ZERO-7B (Zeng et al. 2025), EURUS-2-7B-PRIME (Cui et al. 2025), and OAT-ZERO-7B (Liu et al. 2025); 2) models with code integration, including QWEN2.5-MATH-7B-INSTRUCT-TIR, which enables code execution at inference, TORL-7B (Li, Zou, and Liu 2025) and ZTRL-7B (Mai et al. 2025), which are trained via tool-augmented RL.

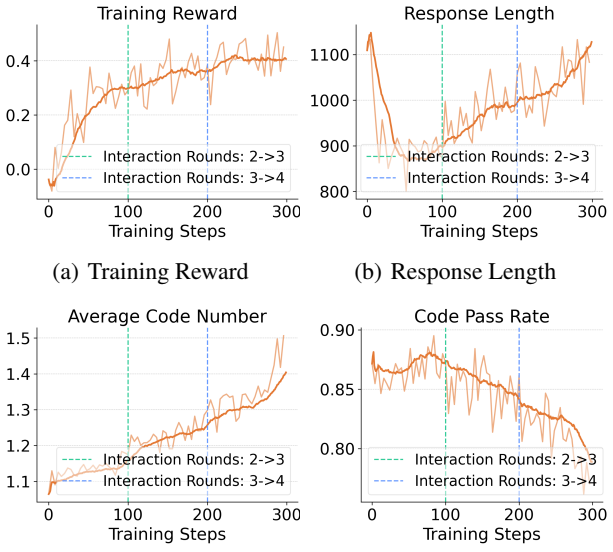
Main Results

ETIR outperforms vanilla and tool-augmented RL methods. As evidenced by Table 1, our method consistently outperforms all baselines, achieving an average accuracy of 52.7%. Compared to vanilla RL methods, ETIR surpasses EURUS-2-7B-PRIME (37.6%) by 15.1 points and OAT-ZERO-7B (39.3%) by 13.4 points. It also exceeds the recent tool-augmented RL method TORL-7B and ZTRL-7B. This improvement stems from both the integration of an external code interpreter, which enhances code-integrated reasoning during RL, and our progressive learning strategy that incrementally develops tool usage capabilities, resulting in significant gains in mathematical reasoning performance.

Models	Code	MATH500	AMC23	AIME2024	AIME2025	OlymMATH	Avg.
QWEN2.5-MATH-7B-INSTRUCT	✗	75.0	45.2	8.12	5.00	3.38	27.3
QWEN2.5-MATH-7B-INSTRUCT-TIR	✔	78.8	59.3	25.8	18.8	18.4	40.2
SIMPLERL-ZERO-7B	✗	75.4	51.1	18.3	6.46	8.75	32.0
EURUS-2-7B-PRIME	✗	81.2	62.8	18.8	15.4	9.94	37.6
OAT-ZERO-7B	✗	79.2	64.2	31.3	10.4	11.3	39.3
TORL-7B	✔	83.8	73.2	37.9	29.2	28.5	50.5
ZTRL-7B	✔	82.0	-	33.3	26.7	12.5	-
ETIR (QWEN2.5-M7)	✔	**86.4**	**74.2**	**42.3**	**29.2**	**31.6**	**52.7**

Table 1: Performance of ETIR and other baselines. For MATH500 , we utilize greedy decoding strategy for evaluation. For the other four benchmarks, we sample 16 responses and report the AVG@16 accuracy.

ETIR achieves superior performance on more challenging benchmarks. Our method demonstrates significantly greater improvements on more challenging benchmarks. On AIME2024, ETIR reaches 42.3%, outperforming EURUS-2-7B-PRIME (18.8%) and OAT-ZERO-7B (31.3%). On AIME2025, it achieves 29.2%, nearly doubling EURUS-2-7B-PRIME (15.4%) and surpassing OAT-ZERO-7B (10.4%) by 18.8 points. On the challenging OlymMATH benchmark, it maintains strong performance at 31.6%. These results underscore how code-integrated reasoning enhances the model's capacity to solve intricate mathematical problems through more sophisticated code solution strategies after tool-augmented RL.



(a) Training Reward (b) Response Length

(c) Average Code Number (d) Code Pass Rate

Figure 3: The learning dynamics of ETIR.

Learning Dynamics Analysis

Training Behavior. As shown in Figures 3(a) and 3(b), ETIR benefits from a gradually increasing interaction budget (2→3→4): rewards rise steadily, while response length first drops—indicating a shift toward concise tool-based reasoning—then increases as the model exploits additional interactions to refine its output.

Code Behavior. The Qwen2.5-Math backbone quickly adapts to code-integrated RL, likely due to prior exposure to code during the pretraining phase (Yang et al. 2024). Figure 3(c) shows that from the early stages, code is used in over 85% of responses, rising to nearly 100% as training progresses. With the increasing interaction budget, this trend indicates that the model gradually adapts to solving problems through code execution rather than relying solely on natural language reasoning. However, the code pass rate exhibits a more complex pattern: it shows an initial upward trend in the first 100 steps, followed by a steady decline. This decline becomes more pronounced as the interaction budget increases—a counterintuitive result that suggests a lower pass rate does not necessarily indicate poorer performance. We will analyze this phenomenon further in the following mechanism analysis section.

Rounds	MATH500	AIME2024	OlymMATH
2 ROUNDS	83.1	38.8	29.4
3 ROUNDS	83.5	39.7	30.1
4 ROUNDS	84.8	40.7	30.6
FROM 2 TO 4	**86.4**	**42.3**	**31.6**

Table 2: Performance of models trained on fixed interaction rounds and progressive increase interaction rounds.

Strategy	MATH500	AIME2024	OlymMATH
ETIR	**86.4**	**42.3**	**31.6**
w/o PMB	84.1	40.2	30.4
w/o ETM	84.0	39.7	30.0

Table 3: Ablation study on other optimization strategies. PMB denotes precise match of interaction boundary; ETM denotes external tool feedback masking.

Ablation Study

We conduct ablation studies to evaluate the effectiveness of each component in our approach.

Models	Mode	AIME2024		AIME2025		OlymMATH		Avg	
		Acc	#Tokens	Acc	#Tokens	Acc	#Tokens	Acc	#Tokens
QWEN3-8B (THINKING-10K)	LCoT	44.6	9,296	32.5	9604	21.9	10,216	33.0	9,705
DISTILL-QWEN-7B	LCoT	51.3	12,974	40.6	13,479	42.2	15,041	**44.7**	13,831
ETIR (QWEN2.5-M7)	CIR	42.3	1,603	29.2	1,843	31.6	1,733	34.4	**1,726**

Table 4: The accuracy and average response length on AIME2024, AIME2025 and OlymMATH between long-CoT and code-integrated reasoning (CIR) models.

As shown in Table 2, models trained with a fixed number of interaction rounds consistently underperform those trained with a progressive schedule that increases the rounds from 2 to 4. Interestingly, even maintaining 4 interaction rounds throughout the entire training process leads to worse performance than gradually increasing them. We attribute this to the progressive schedule's ability to help the model gradually adapt to increasing interaction demands, mitigating unstable reasoning behaviors caused by an overly generous interaction budget in early stages and reducing the risk of premature convergence to rigid interaction patterns.

In addition, Table 3 shows that the precise match of interaction boundaries and external tool feedback masking also prove to be effective. Removing these strategies during training leads to a decrease in model performance.

Mechanism of Code-Integrated Reasoning

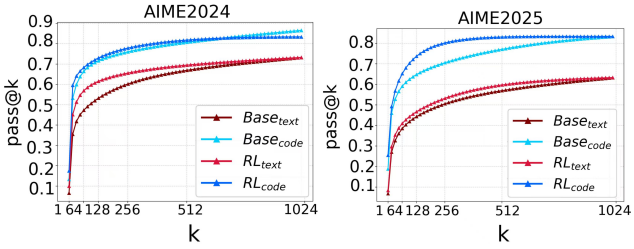
While code integration aids math reasoning, its underlying benefits and how it boosts performance remain underexplored. This section analyzes the key mechanisms through which code-integrated reasoning, especially after tool-augmented RL, improves model performance.

Impact on Model's Capability Boundaries

RL guides models to improve via reward signals but mainly relies on self-generated data, limiting capacity expansion (Gandhi et al. 2025; Wang et al. 2025a). In contrast, code-integrated reasoning uses external code interpreters to execute generated code, differing fundamentally from standard text-based reasoning. We therefore hypothesize that code integration can significantly extend model capabilities and conduct an empirical study to verify this.

Evaluation settings. To investigate the impact of code-integrated reasoning on the model's capability boundaries, we test four baseline models, which are all derived from QWEN2.5-MATH-7B, on the AIME2024 and AIME2025 benchmarks, using the PASS@k metric. For all experiments, we use a temperature of 0.6 and a top-p value of 0.95, allowing for a maximum generation of 16,384 tokens. The four baselines are as follows:

- $Base_{text}$: The base model, utilizing standard text-based reasoning.
- $Base_{code}$: The base model, utilizing code-integrated reasoning via the direct prompt.
- RL_{text}: The model trained with vanilla RL, utilizing text-based reasoning.
- RL_{code}: The model trained with tool-augmented RL, utilizing code-integrated reasoning.



(a) AIME2024 (b) AIME2025

Figure 4: PASS@k accuracy on AIME2024 and AIME2025.

Code-integrated reasoning can expand model's capability boundaries. As illustrated in Figure 4, Code-integrated reasoning significantly improves PASS@K performance over text-based reasoning, indicating an expanded capability boundary. However, consistent with prior findings (Yue et al. 2025a), as k increases, RL-trained models are eventually matched or outperformed by their non-RL counterparts, which reveals that RL alone does not effectively enhance knowledge acquisition.

Comparison between CIR and Long-CoT

To compare CIR with standard long-CoT reasoning (Yeo et al. 2025) without external tools, we use two reference models: QWEN3-8B (THINKING-10K) and DEEPSEEK-R1-DISTILL-QWEN-7B, with QWEN3-8B limited to 10K tokens to maintain evaluation efficiency. Table 4 reports their accuracy and response lengths.

As shown in Table 4, our model achieves similar accuracy with much shorter responses—reaching nearly 80% of DEEPSEEK-R1-DISTILL-QWEN-7B's accuracy using under 15% of its length, and matching QWEN3-8B (THINKING-10K) with less than 20%. This highlights the efficiency of code-integrated reasoning.

To investigate why CIR yields more concise responses, we manually inspect the outputs and summarize the findings: 1) CIR tends to start with a brief solution overview and utilize the structured, deterministic nature of programming to execute precise, unambiguous computations, avoiding lengthy reasoning; 2) Long-CoT reasoning mimics human-like thought, involving iterative reflection, hypothesis revision, and path exploration. This often leads to longer responses as the model considers multiple solution paths.

Effects of Non-Executable Code

Translating problem-solving steps into code for external execution can boost accuracy, but the effect of non-executable code on performance is still unclear.

To investigate this, we analyze the code pass rate for correct and incorrect responses from our tool-augmented RL model on 60 AIME2024–2025 problems, with 16 samples per problem. As shown in Table 5, about one-third of correct responses contain non-executable code, while over one-third of incorrect responses produce fully executable code. From these results and further case analysis, we draw the following conclusions.

	ACN	FuPR	NR	FiPR
CORRECT (345)	1.47	60.6%	39.4%	65.5%
INCORRECT (615)	2.11	36.0%	64.0%	40.2%

Table 5: *Average Code Number*, *Full Pass Rate*, *Non-execution Rate* and *Final Pass Rate* for correct and incorrect responses. *ACN* (average code number), *FuPR* (ratio of responses with all code running successfully), *NR* (ratio of responses containing non-executable code), and *FiPR* (ratio of responses whose final code runs without error). Numbers in brackets indicate response counts.

• **Executable but logically flawed code may hinder further improvements in model performance.** While executable code can offer seemingly valid feedback, it may prematurely halt the reasoning process when the underlying logic is flawed. As shown in Table 5, 36.0% of incorrect responses stem from this issue: although the code runs successfully, its flawed logic misleads the model into accepting incorrect (see in Figure 5). This reveals a key limitation: executable code does not ensure response quality. Actually, successfully executed code can be especially misleading because models often accept its feedback without any scrutiny.

• **Error feedback from non-executable code can aid code-integrated reasoning.** Executing non-executable code yields error feedback that the model cannot produce on its own, prompting reflection and code revision. This process helps the model develop executable and logically correct solutions, improving accuracy. As Table 5 shows, 39.4% of correct responses involve non-executable code, indicating its contribution to valid reasoning. Additionally, the declining pass rate in Figures 3(c) and 3(d) suggests that increased interaction with the code interpreter during training reflects more code revisions. This is further supported by the difference between *Full Pass Rate* (60.6%) and *Final Pass Rate* (65.5%) in Table 5, where about 5% of responses correspond to successfully revised code leading to correct answers.

Analysis of Performance Gain Differences

While code-integrated reasoning can enhance mathematical reasoning, its benefits may vary across different problem types. To better understand it, we examine how the advantages of code-integrated reasoning differ among various categories of mathematical problems. We evaluate three models, $Base_{text}$, $Base_{tool}$ and RL_{tool}, on four main categories of

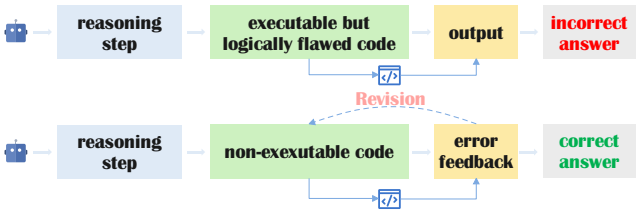
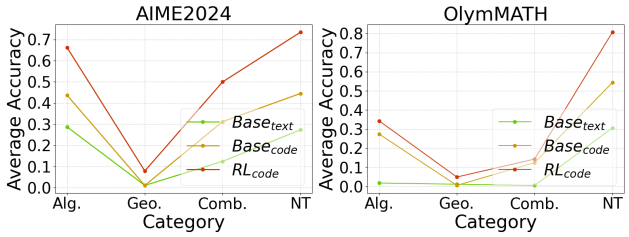


Figure 5: Flawed but executable code misleads the model, whereas non-executable code forces revision and leads to correct solutions.



(a) AIME2024 (b) OlymMATH

Figure 6: Accuracy of different categories on AIME2024 and OlymMATH.

mathematical problems (algebra, geometry, combinatorics, and number theory) from AIME2024 and OlymMATH.

Figure 6 shows that the effectiveness of code integration varies significantly across problem types. It leads to substantial accuracy gains in algebra, number theory, and combinatorics by offloading complex computations and mitigating errors from lengthy reasoning, which is further amplified by RL. In contrast, geometry exhibits only marginal improvements. This is likely because geometry relies more on spatial and conceptual reasoning, where correct problem framing is more critical than computation, limiting the utility of code execution when the underlying logic is flawed.

Conclusion

In this paper, we analyze key factors causing the instability of tool-augmented RL, including noisy interactions, distributional shift, and response homogenization. Based on these findings, we propose ETIR, a novel method to introduce progressive learning into tool-augmented RL for the first time. ETIR gradually builds tool-use capabilities, enabling more efficient and adaptive model-environment interaction, and achieves superior performance over both vanilla RL and existing tool-augmented baselines on challenging mathematical benchmarks. We further investigate the mechanism behind code-integrated reasoning, an aspect often overlooked in prior work, and demonstrate it extends the model's reasoning capacity and promotes more concise, structured problem-solving, which might offer useful insights for subsequent research. Although our method is implemented on code execution, it is inherently extensible to other external tools. Our future work will focus on generalizing this framework to further enhance the model's capabilities.

Acknowledgements

This paper was partially supported by the National Natural Science Foundation of China No. 92470205 and 62222215. Xin Zhao is the corresponding author.

References

Chen, Z.; Min, Y.; Zhang, B.; Chen, J.; Jiang, J.; Cheng, D.; Zhao, W. X.; Liu, Z.; Miao, X.; Lu, Y.; Fang, L.; Wang, Z.; and Wen, J.-R. 2025. An Empirical Study on Eliciting and Improving R1-like Reasoning Models. arXiv:2503.04548.

Chen, Z.; Zhou, K.; Zhang, B.; Gong, Z.; Zhao, W. X.; and Wen, J.-R. 2023. ChatCoT: Tool-Augmented Chain-of-Thought Reasoning on Chat-based Large Language Models. arXiv:2305.14323.

Cheng, D.; Huang, S.; Zhu, X.; Dai, B.; Zhao, W. X.; Zhang, Z.; and Wei, F. 2025. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758*.

Cui, G.; Yuan, L.; Wang, Z.; Wang, H.; Li, W.; He, B.; Fan, Y.; Yu, T.; Xu, Q.; Chen, W.; Yuan, J.; Chen, H.; Zhang, K.; Lv, X.; Wang, S.; Yao, Y.; Han, X.; Peng, H.; Cheng, Y.; Liu, Z.; Sun, M.; Zhou, B.; and Ding, N. 2025. Process Reinforcement through Implicit Rewards. arXiv:2502.01456.

DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; Zhang, X.; Yu, X.; Wu, Y.; Wu, Z. F.; Gou, Z.; Shao, Z.; Li, Z.; Gao, Z.; Liu, A.; Xue, B.; Wang, B.; Wu, B.; Feng, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; Dai, D.; Chen, D.; Ji, D.; Li, E.; Lin, F.; Dai, F.; Luo, F.; Hao, G.; Chen, G.; Li, G.; Zhang, H.; Bao, H.; Xu, H.; Wang, H.; Ding, H.; Xin, H.; Gao, H.; Qu, H.; Li, H.; Guo, J.; Li, J.; Wang, J.; Chen, J.; Yuan, J.; Qiu, J.; Li, J.; Cai, J. L.; Ni, J.; Liang, J.; Chen, J.; Dong, K.; Hu, K.; Gao, K.; Guan, K.; Huang, K.; Yu, K.; Wang, L.; Zhang, L.; Zhao, L.; Wang, L.; Zhang, L.; Xu, L.; Xia, L.; Zhang, M.; Zhang, M.; Tang, M.; Li, M.; Wang, M.; Li, M.; Tian, N.; Huang, P.; Zhang, P.; Wang, Q.; Chen, Q.; Du, Q.; Ge, R.; Zhang, R.; Pan, R.; Wang, R.; Chen, R. J.; Jin, R. L.; Chen, R.; Lu, S.; Zhou, S.; Chen, S.; Ye, S.; Wang, S.; Yu, S.; Zhou, S.; Pan, S.; Li, S. S.; Zhou, S.; Wu, S.; Ye, S.; Yun, T.; Pei, T.; Sun, T.; Wang, T.; Zeng, W.; Zhao, W.; Liu, W.; Liang, W.; Gao, W.; Yu, W.; Zhang, W.; Xiao, W. L.; An, W.; Liu, X.; Wang, X.; Chen, X.; Nie, X.; Cheng, X.; Liu, X.; Xie, X.; Liu, X.; Yang, X.; Li, X.; Su, X.; Lin, X.; Li, X. Q.; Jin, X.; Shen, X.; Chen, X.; Sun, X.; Wang, X.; Song, X.; Zhou, X.; Wang, X.; Shan, X.; Li, Y. K.; Wang, Y. Q.; Wei, Y. X.; Zhang, Y.; Xu, Y.; Li, Y.; Zhao, Y.; Sun, Y.; Wang, Y.; Yu, Y.; Zhang, Y.; Shi, Y.; Xiong, Y.; He, Y.; Piao, Y.; Wang, Y.; Tan, Y.; Ma, Y.; Liu, Y.; Guo, Y.; Ou, Y.; Wang, Y.; Gong, Y.; Zou, Y.; He, Y.; Xiong, Y.; Luo, Y.; You, Y.; Liu, Y.; Zhou, Y.; Zhu, Y. X.; Xu, Y.; Huang, Y.; Li, Y.; Zheng, Y.; Zhu, Y.; Ma, Y.; Tang, Y.; Zha, Y.; Yan, Y.; Ren, Z. Z.; Ren, Z.; Sha, Z.; Fu, Z.; Xu, Z.; Xie, Z.; Zhang, Z.; Hao, Z.; Ma, Z.; Yan, Z.; Wu, Z.; Gu, Z.; Zhu, Z.; Liu, Z.; Li, Z.; Xie, Z.; Song, Z.; Pan, Z.; Huang, Z.; Xu, Z.; Zhang, Z.; and Zhang, Z. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948.

Dong, G.; Chen, Y.; Li, X.; Jin, J.; Qian, H.; Zhu, Y.; Mao, H.; Zhou, G.; Dou, Z.; and Wen, J.-R. 2025. Tool-Star: Empowering LLM-Brained Multi-Tool Reasoner via Reinforcement Learning. arXiv:2505.16410.

Feng, J.; Huang, S.; Qu, X.; Zhang, G.; Qin, Y.; Zhong, B.; Jiang, C.; Chi, J.; and Zhong, W. 2025. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. arXiv:2504.11536.

Gandhi, K.; Chakravarthy, A.; Singh, A.; Lile, N.; and Goodman, N. D. 2025. Cognitive Behaviors that Enable Self-Improving Reasoners, or, Four Habits of Highly Effective STaRs. arXiv:2503.01307.

Goldie, A.; Mirhoseini, A.; Zhou, H.; Cai, I.; and Manning, C. D. 2025. Synthetic Data Generation & Multi-Step RL for Reasoning & Tool Use. arXiv:2504.04736.

Gou, Z.; Shao, Z.; Gong, Y.; Shen, Y.; Yang, Y.; Huang, M.; Duan, N.; and Chen, W. 2024. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. arXiv:2309.17452.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. arXiv:2103.03874.

Hu, J.; Liu, J. K.; Xu, H.; and Shen, W. 2025. REINFORCE++: An Efficient RLHF Algorithm with Robustness to Both Prompt and Reward Models. arXiv:2501.03262.

Li, C.; Xue, M.; Zhang, Z.; Yang, J.; Zhang, B.; Wang, X.; Yu, B.; Hui, B.; Lin, J.; and Liu, D. 2025a. START: Self-taught Reasoner with Tools. arXiv:2503.04625.

Li, X.; Dong, G.; Jin, J.; Zhang, Y.; Zhou, Y.; Zhu, Y.; Zhang, P.; and Dou, Z. 2025b. Search-o1: Agentic Search-Enhanced Large Reasoning Models. arXiv:2501.05366.

Li, X.; Zou, H.; and Liu, P. 2025. ToRL: Scaling Tool-Integrated RL. arXiv:2503.23383.

Liu, Z.; Chen, C.; Li, W.; Qi, P.; Pang, T.; Du, C.; Lee, W. S.; and Lin, M. 2025. Understanding R1-Zero-Like Training: A Critical Perspective. arXiv:2503.20783.

Lu, P.; Chen, B.; Liu, S.; Thapa, R.; Boen, J.; and Zou, J. 2025. OctoTools: An Agentic Framework with Extensible Tools for Complex Reasoning. arXiv:2502.11271.

Mai, X.; Xu, H.; W, X.; Wang, W.; Hu, J.; Zhang, Y.; and Zhang, W. 2025. Agent RL Scaling Law: Agent RL with Spontaneous Code Execution for Mathematical Problem Solving. arXiv:2505.07773.

MiniMax; :; Chen, A.; Li, A.; Gong, B.; Jiang, B.; Fei, B.; Yang, B.; Shan, B.; Yu, C.; Wang, C.; Zhu, C.; Xiao, C.; Du, C.; Zhang, C.; Qiao, C.; Zhang, C.; Du, C.; Guo, C.; Chen, D.; Ding, D.; Sun, D.; Li, D.; Jiao, E.; Zhou, H.; Zhang, H.; Ding, H.; Sun, H.; Feng, H.; Cai, H.; Zhu, H.; Sun, J.; Zhuang, J.; Cai, J.; Song, J.; Zhu, J.; Li, J.; Tian, J.; Liu, J.; Xu, J.; Yan, J.; Liu, J.; He, J.; Feng, K.; Yang, K.; Xiao, K.; Han, L.; Wang, L.; Yu, L.; Feng, L.; Li, L.; Zheng, L.; Du, L.; Yang, L.; Zeng, L.; Yu, M.; Tao, M.; Chi, M.; Zhang, M.; Lin, M.; Hu, N.; Di, N.; Gao, P.; Li, P.; Zhao, P.; Ren, Q.; Xu, Q.; Li, Q.; Wang, Q.; Tian, R.; Leng, R.; Chen, S.; Chen, S.; Shi, S.; Weng, S.; Guan, S.; Yu, S.; Li, S.; Zhu, S.; Li, T.; Cai,

T.; Liang, T.; Cheng, W.; Kong, W.; Li, W.; Chen, X.; Song, X.; Luo, X.; Su, X.; Li, X.; Han, X.; Hou, X.; Lu, X.; Zou, X.; Shen, X.; Gong, Y.; Ma, Y.; Wang, Y.; Shi, Y.; Zhong, Y.; Duan, Y.; Fu, Y.; Hu, Y.; Gao, Y.; Fan, Y.; Yang, Y.; Li, Y.; Hu, Y.; Huang, Y.; Li, Y.; Xu, Y.; Mao, Y.; Shi, Y.; Wenren, Y.; Li, Z.; Li, Z.; Tian, Z.; Zhu, Z.; Fan, Z.; Wu, Z.; Xu, Z.; Yu, Z.; Lyu, Z.; Jiang, Z.; Gao, Z.; Wu, Z.; Song, Z.; and Sun, Z. 2025. MiniMax-M1: Scaling Test-Time Compute Efficiently with Lightning Attention. arXiv:2506.13585.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C. L.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; Schulman, J.; Hilton, J.; Kelton, F.; Miller, L.; Simens, M.; Askell, A.; Welinder, P.; Christiano, P.; Leike, J.; and Lowe, R. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155.

Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y. K.; Wu, Y.; and Guo, D. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300.

Sheng, G.; Zhang, C.; Ye, Z.; Wu, X.; Zhang, W.; Zhang, R.; Peng, Y.; Lin, H.; and Wu, C. 2025. HybridFlow: A Flexible and Efficient RLHF Framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, 1279–1297. ACM.

Song, H.; Jiang, J.; Min, Y.; Chen, J.; Chen, Z.; Zhao, W. X.; Fang, L.; and Wen, J.-R. 2025a. R1-Searcher: Incentivizing the Search Capability in LLMs via Reinforcement Learning. arXiv:2503.05592.

Song, H.; Jiang, J.; Tian, W.; Chen, Z.; Wu, Y.; Zhao, J.; Min, Y.; Zhao, W. X.; Fang, L.; and Wen, J.-R. 2025b. R1-Searcher++: Incentivizing the Dynamic Knowledge Acquisition of LLMs via Reinforcement Learning. arXiv:2505.17005.

Sun, H.; Min, Y.; Chen, Z.; Zhao, W. X.; Fang, L.; Liu, Z.; Wang, Z.; and Wen, J.-R. 2025a. Challenging the Boundaries of Reasoning: An Olympiad-Level Math Benchmark for Large Language Models. arXiv:2503.21380.

Sun, S.; Song, H.; Wang, Y.; Ren, R.; Jiang, J.; Zhang, J.; Bai, F.; Deng, J.; Zhao, W. X.; Liu, Z.; Fang, L.; Wang, Z.; and Wen, J.-R. 2025b. SimpleDeepSearcher: Deep Information Seeking via Web-Powered Reasoning Trajectory Synthesis. arXiv:2505.16834.

Team, K.; Du, A.; Gao, B.; Xing, B.; Jiang, C.; Chen, C.; Li, C.; Xiao, C.; Du, C.; Liao, C.; et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Wang, H.; Xue, B.; Zhou, B.; Zhang, T.; Wang, C.; Wang, H.; Chen, G.; and fai Wong, K. 2025a. Self-DC: When to Reason and When to Act? Self Divide-and-Conquer for Compositional Unknown Questions. arXiv:2402.13514.

Wang, Z.; Wang, K.; Wang, Q.; Zhang, P.; Li, L.; Yang, Z.; Jin, X.; Yu, K.; Nguyen, M. N.; Liu, L.; Gottlieb, E.; Lu, Y.; Cho, K.; Wu, J.; Fei-Fei, L.; Wang, L.; Choi, Y.; and Li, M. 2025b. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning. arXiv:2504.20073.

Wu, F.; Xuan, W.; Lu, X.; Harchaoui, Z.; and Choi, Y. 2025. The Invisible Leash: Why RLVR May Not Escape Its Origin. arXiv:2507.14843.

Yang, A.; Zhang, B.; Hui, B.; Gao, B.; Yu, B.; Li, C.; Liu, D.; Tu, J.; Zhou, J.; Lin, J.; Lu, K.; Xue, M.; Lin, R.; Liu, T.; Ren, X.; and Zhang, Z. 2024. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. arXiv:2409.12122.

Yeo, E.; Tong, Y.; Niu, M.; Neubig, G.; and Yue, X. 2025. Demystifying Long Chain-of-Thought Reasoning in LLMs. arXiv:2502.03373.

Yu, Q.; Zhang, Z.; Zhu, R.; Yuan, Y.; Zuo, X.; Yue, Y.; Dai, W.; Fan, T.; Liu, G.; Liu, L.; Liu, X.; Lin, H.; Lin, Z.; Ma, B.; Sheng, G.; Tong, Y.; Zhang, C.; Zhang, M.; Zhang, W.; Zhu, H.; Zhu, J.; Chen, J.; Chen, J.; Wang, C.; Yu, H.; Song, Y.; Wei, X.; Zhou, H.; Liu, J.; Ma, W.-Y.; Zhang, Y.-Q.; Yan, L.; Qiao, M.; Wu, Y.; and Wang, M. 2025. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. arXiv:2503.14476.

Yuan, Y.; Yue, Y.; Zhu, R.; Fan, T.; and Yan, L. 2025. What's Behind PPO's Collapse in Long-CoT? Value Optimization Holds the Secret. arXiv:2503.01491.

Yue, Y.; Chen, Z.; Lu, R.; Zhao, A.; Wang, Z.; Yue, Y.; Song, S.; and Huang, G. 2025a. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? arXiv:2504.13837.

Yue, Y.; Yuan, Y.; Yu, Q.; Zuo, X.; Zhu, R.; Xu, W.; Chen, J.; Wang, C.; Fan, T.; Du, Z.; Wei, X.; Yu, X.; Liu, G.; Liu, J.; Liu, L.; Lin, H.; Lin, Z.; Ma, B.; Zhang, C.; Zhang, M.; Zhang, W.; Zhu, H.; Zhang, R.; Liu, X.; Wang, M.; Wu, Y.; and Yan, L. 2025b. VAPO: Efficient and Reliable Reinforcement Learning for Advanced Reasoning Tasks. arXiv:2504.05118.

Zeng, W.; Huang, Y.; Liu, Q.; Liu, W.; He, K.; Ma, Z.; and He, J. 2025. SimpleRL-Zoo: Investigating and Taming Zero Reinforcement Learning for Open Base Models in the Wild. arXiv:2503.18892.

Zhang, S.; Dong, Y.; Zhang, J.; Kautz, J.; Catanzaro, B.; Tao, A.; Wu, Q.; Yu, Z.; and Liu, G. 2025a. Nemotron-Research-Tool-N1: Exploring Tool-Using Language Models with Reinforced Reasoning. arXiv:2505.00024.

Zhang, X.; Wang, J.; Cheng, Z.; Zhuang, W.; Lin, Z.; Zhang, M.; Wang, S.; Cui, Y.; Wang, C.; Peng, J.; Jiang, S.; Kuang, S.; Yin, S.; Wen, C.; Zhang, H.; Chen, B.; and Yu, B. 2025b. SRPO: A Cross-Domain Implementation of Large-Scale Reinforcement Learning on LLM. arXiv:2504.14286.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; Du, Y.; Yang, C.; Chen, Y.; Chen, Z.; Jiang, J.; Ren, R.; Li, Y.; Tang, X.; Liu, Z.; Liu, P.; Nie, J.-Y.; and Wen, J.-R. 2025. A Survey of Large Language Models. arXiv:2303.18223.