

SimpleTIR: End-to-End Reinforcement Learning for Multi-Turn Tool-Integrated Reasoning

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Large Language Models (LLMs) can enhance their reasoning by interacting with
2 external tools, a paradigm known as Tool-Integrated Reasoning (TIR). However,
3 extending TIR to multi-turn settings using Reinforcement Learning (RL) often
4 exhibits training instability and degraded performance. We attribute the instability
5 to harmful negative samples resulting from distributional drift and compounding
6 errors induced by using external tool outputs during multi-turn rollout. To address
7 this issue, we introduce SimpleTIR, a simple method that stabilizes multi-turn
8 TIR training via filtering out trajectories with “void turns”, i.e., turns that yield
9 neither a code block nor a final answer. Specifically, we remove those trajec-
10 tories from the policy update to block harmful gradients, while retaining them
11 in advantage estimation to keep the estimate unbiased. Extensive experiments
12 show that SimpleTIR effectively mitigates gradient norm explosion and stabilizes
13 multi-turn RL training from base models. It achieves state-of-the-art performance
14 on challenging math reasoning benchmarks, including an AIME24 score of 50.5
15 starting from the Qwen2.5-7B base model. SimpleTIR also promotes more diverse
16 reasoning behaviors such as self-correction and cross-validation, outperforming
17 prior methods trained from stronger instruction-tuned models.

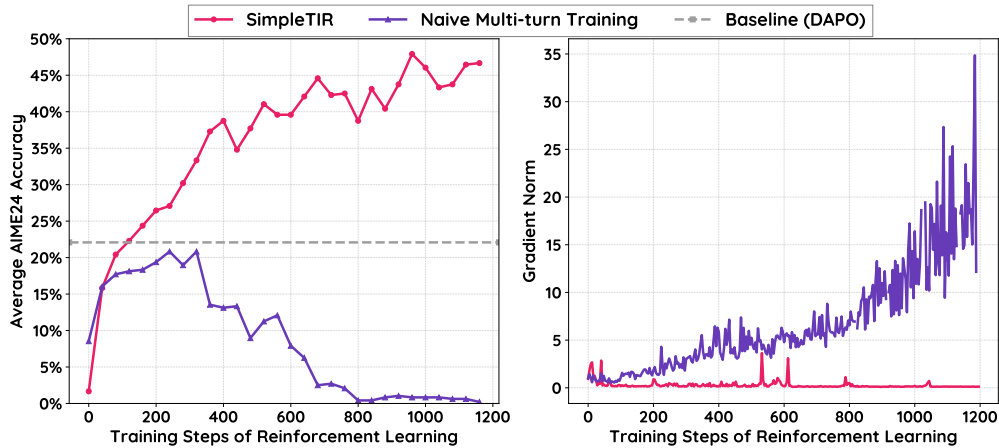


Figure 1: Starting from Qwen2.5-7B base model, The training dynamics of SimpleTIR are highly stable, and it clearly outperforms the baseline method without TIR (DAPO). The gradient norm remains well-behaved with almost no spikes. In contrast, Naive Multi-turn Training not only suffers from unstable dynamics and catastrophic gradient norm explosions, but also fails to match the performance of the baseline without TIR.

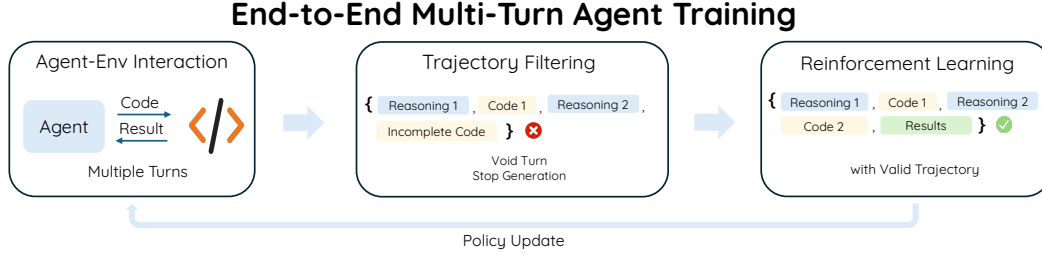


Figure 2: The overview of SimpleTIR. During policy update, SimpleTIR filters out trajectories containing void turns, i.e., an LLM response that fails to produce a complete code block or a final answer.

18 1 Introduction

19 Training Large Language Models (LLMs) to perform *multi-turn* Tool-Integrated Reasoning (TIR) is
 20 one of the most promising frontiers in Reinforcement Learning (RL). At each turn, LLMs iteratively
 21 reason, generate code, execute it, and use the output to help reasoning in the next turn. TIR addresses
 22 LLMs’ inherent limitations such as limited calculation accuracy and knowledge cutoff. For example,
 23 LLMs can perform exact mathematical computation via a Python interpreter or retrieve up-to-date
 24 information via search engines. Despite its clear potential, training LLMs for multi-turn TIR remains
 25 highly challenging, as the process often suffers from instability and gradient explosion [1, 2, 3, 4].

26 To probe training instability, we start from a single-turn TIR setup in which LLMs are allowed
 27 to generate only one response (text reasoning plus an optional code block). As shown in Fig. 3,
 28 *single-turn TIR training is surprisingly stable*, while naive multi-turn TIR training suffers from
 29 gradient explosion and performance drop. We attribute this gap to the emergence and propagation of
 30 *low-probability tokens* induced by using out-of-distribution (OOD) tool outputs as model inputs for
 31 subsequent turns. Specifically, when the model’s response is concatenated with external tool output,
 32 the resulting next turn inputs can depart from the model distribution. The subsequent turns inherit
 33 this drift and accumulate low-probability tokens (Fig. 4), eventually collapsing RL training. We argue
 34 that multi-turn training instability mainly stems from the following two issues: (i) harmful gradients
 35 from low-probability tokens, which occur more often in multi-turn settings because external tool
 36 outputs can push inputs off distribution, making the model sample low-probability tokens that cause
 37 gradient spikes during RL updates, and (ii) problematic multi-turn credit assignment, where errors in
 38 later turns cause the correct reasoning in earlier turns to be mistakenly assigned a negative reward.
 39 Fundamentally, both issues can be understood as the existence of **harmful negative samples in the**
 40 **training process**.

41 We propose SimpleTIR, a simple yet effective algorithm that stabilizes multi-turn TIR training. The
 42 key idea is to leverage *void turns* to distinguish helpful negative samples from harmful ones. A void
 43 turn is defined as an LLM response that fails to produce a complete code block or a final answer.
 44 Typical examples include partial code blocks, repetitive text, or incomplete responses caused by the
 45 model prematurely sampling an end-of-sequence (eos) token. The core insight of SimpleTIR is to
 46 **filter out trajectories containing void turns**. Specifically, SimpleTIR excludes these trajectories
 47 when computing the policy loss, thus blocking harmful gradients from propagating. Meanwhile, it still
 48 incorporates them into advantage estimation, so that the estimate is unbiased. This filtering strategy
 49 effectively mitigates instability caused by harmful negative samples. More importantly, SimpleTIR is
 50 general and plug-and-play. It requires only minimal modifications and can be seamlessly integrated
 51 into existing training frameworks to improve stability and performance with almost no extra cost.

52 To validate the effectiveness of SimpleTIR, we conduct comprehensive experiments on challenging
 53 math reasoning tasks across various settings. SimpleTIR exhibits state-of-the-art multi-turn TIR
 54 performance starting from base models, achieving an AIME24 score of 50.5. Our ablation studies
 55 further highlight the importance of void turn filtering, demonstrating that excluding trajectories with
 56 void turns is the key to stabilizing multi-turn TIR training and unlocking performance improvements.
 57 It directly addresses the instability that hinders naive multi-turn approaches. Finally, we evaluate the
 58 benefits of Zero RL for multi-turn TIR training. Unlike approaches that first “cold start” LLMs before
 59 RL [5], SimpleTIR allows LLMs to explore more novel and diverse multi-turn reasoning patterns,
 60 including cross-validation, progressive reasoning, and self-correction.

61 2 Preliminaries

62 2.1 Training LLMs with RL

63 We model the auto-regressive LLM generation process as a Markov Decision Process (MDP) $<$
64 $\mathcal{S}, \mathcal{A}, T, R, q, \gamma >$, where the state space $\mathcal{S}$ includes possible prefixes when generating the next token
65 and the action space $\mathcal{A}$ includes all possible tokens. The transition T appends the new tokens into
66 the prefixes. R is the reward function and q is the initial prompt distribution. The discount factor
67 γ is usually set to 1 in LLM generation. Instead of applying the widely-employed Proximal Policy
68 Optimization (PPO) [6], Group Relative Policy Optimization (GRPO) [7] is generally more preferable
69 due to its efficient computation of trajectory advantage

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\{r_i\}_{i=1}^G)}{F_{\text{norm}}(\{r_i\}_{i=1}^G)}, \quad (1)$$

70 where $F_{\text{norm}} = \text{std}$ in the original GRPO and $F_{\text{norm}} = 1$ may also be used to reduce difficulty bias.
71 The policy loss in GRPO is

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,a) \sim \mathcal{D}, \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left(\min(\rho_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(\rho_{i,t}(\theta), 1 - \varepsilon, 1 + \varepsilon) \hat{A}_{i,t}) - \beta D_{\text{KL}}(\pi_{\theta} || \pi_{\text{ref}}) \right) \right], \quad (2)$$

72 where $\rho_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q, o_{i,<t})}$ and β is often set to 0 in Zero RL.

73 2.2 Tool Integrated Reasoning

74 In our TIR setting, each LLM response is truncated after a complete code block. A code interpreter is
75 available to detect and execute code blocks in LLM responses. The interpreter feedback f , which
76 is either standard output or standard error, will be appended after the LLM response l . The whole
77 response at turn k will become $o = (q, l_0, f_0, l_1, f_1, \dots, l_k, f_k)$. From the MDP perspective, when
78 complete code blocks are generated, the transition function T appends not only the new token,
79 but also the code execution result, to the original prefix. We do not consider format rewards or
80 intermediate feedback in multi-turn TIR. The outcome reward is provided after a complete response
81 o is generated.

82 3 Method

83 We begin by diagnosing a core source of instability in multi-turn TIR: the emergence of low-
84 probability tokens. We show how such tokens drive gradient explosions and misaligned credit
85 assignment during Zero-RL training. Building on this analysis, we propose a simple but effective
86 trajectory filtering scheme that stabilizes training while preserving multi-turn behaviors. This section
87 presents the phenomenon, the analysis, and the algorithm, followed by practical training details.

88 3.1 Emergence of Low-probability Tokens in Multi-turn TIR Trajectories

89 Given the notorious instability in training multi-turn TIR [1, 2, 3], we begin with a minimal single-turn
90 TIR setting where the model produces exactly one response, including text reasoning and an optional
91 code block. A `final_answer` function is provided in each code block for printing the indicated
92 answer. As shown in Fig. 3, naive single-turn training is notably more stable than its multi-turn
93 counterpart, exhibiting higher performance and fewer gradient spikes. The key difference is that in
94 multi-turn TIR, the response l_k and the tool feedback f_k in turn k are concatenated into the input
95 for turn $k + 1$. Because the feedback is produced by an external code interpreter rather than the
96 model itself, it can deviate substantially from the model’s own distribution. With such OOD inputs,
97 the model outputs may deviate from pretrained patterns and become highly stochastic, exhibiting
98 unnaturally low probabilities on selected tokens.

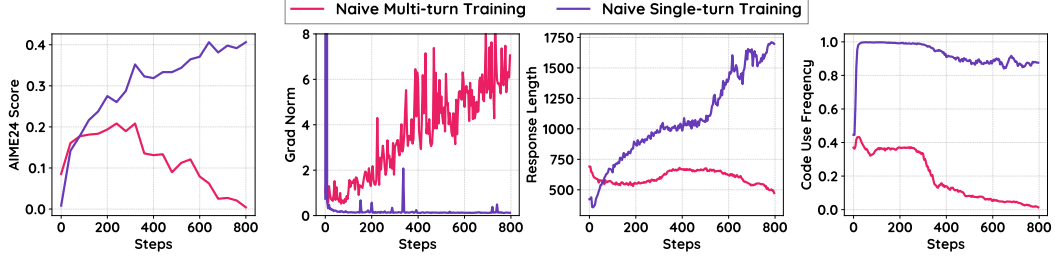


Figure 3: Training statistics comparisons between naive **single-turn** and **multi-turn** TIR. Single-turn TIR trains smoothly with reasonably good performance.

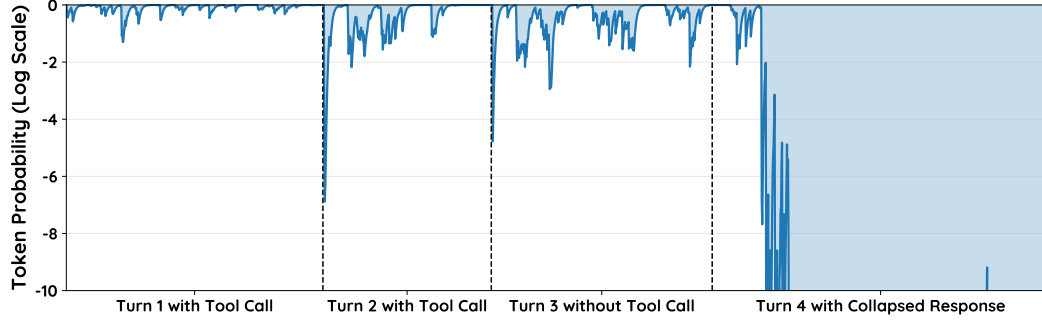


Figure 4: Visualization of token probabilities in a response generated by multi-turn TIR. The y-axis is in log scale and the x-axis is separated by different turns of generations.

To verify this, a case study is conducted on a response generated under multi-turn TIR. As illustrated by Fig. 4, the tool feedback of Turn 1 and Turn 2 indeed contain extremely low token probabilities and strong OOD characteristics. Similar observations have been reported in previous TIR research [8, 2] and can be handled by masking the policy loss on tool feedback. However, even if tool outputs themselves are masked, later model responses can still be degraded. The probabilities of tokens before tool calls, i.e., in Turn 1, remain relatively high, yet multiple low-probability segments emerge in Turn 2 and Turn 3. This distributional drift compounds across turns and culminates in a collapsed response with extremely low token probabilities in Turn 4.

3.2 Low-probability Tokens Compromise Zero RL Training

We identify a problematic pattern in multi-turn TIR trajectories: existence of low-probability tokens. We analyze in the following subsection two consequences of such pattern in Zero RL training. This explains the training instability and inefficiency of naive multi-turn TIR, as demonstrated in Fig. 3.

Gradient Explosion Fig. 3 reveals a dominant failure mode in multi-turn TIR training: gradient explosion. Motivated by Li [9], we make theoretical analysis on the policy gradient with respect to softmax logits $\mathbf{z}$ and connect gradient explosion to the issue of low-probability tokens in Sec. 3.1. The gradient on the policy parameters θ depends on the specific network architecture and is not considered here. The following theorem describes several influencing factors when computing policy gradient.

Proposition 1. Consider a certain token c in a LLM-generated response o with prompt q . The gradient of the GRPO objective with respect to the logits $\mathbf{z}$ at the position of c is

$$\|\nabla_{\mathbf{z}} \mathcal{J}_{\text{GRPO}}\|_2 = \frac{1}{|o|} \cdot \frac{\pi_{\theta}(c|q, o)}{\pi_{\theta_{\text{old}}}(c|q, o)} g_c |\hat{A}_c| \sqrt{1 - 2P(c) + \sum_j P(j)^2}, \quad (3)$$

where $|\hat{A}_c|$ is the absolute value of advantage of token c , P is the probability vector at the position of c with $P(j) = \pi_{\theta}(j|q, o)$, $j \in \mathcal{A}$, and g is the gating function

$$g = \mathbf{1}\{\hat{A}_c \geq 0, \frac{\pi_{\theta}(c|q, o)}{\pi_{\theta_{\text{old}}}(c|q, o)} \leq 1 + \varepsilon\} + \mathbf{1}\{\hat{A}_c < 0, \frac{\pi_{\theta}(c|q, o)}{\pi_{\theta_{\text{old}}}(c|q, o)} \geq 1 - \varepsilon\}.$$

According to Thm. 1, the scale of gradients on logits is primarily determined by the probability ratio $\frac{\pi_\theta(c|q,o)}{\pi_{\theta_{\text{old}}}(c|q,o)}$ and the probability expression $\sqrt{1 - 2P(c) + \sum_j P(j)^2}$. We argue that both terms can contribute to unhealthy policy gradients in the existence of low-probability tokens as demonstrated in Sec. 3.1:

- Constantly high gradient norm due to $\sqrt{1 - 2P(c) + \sum_j P(j)^2}$: As shown in Fig. 4, the model may place very low probability on the selected token c so that $1 - 2P(c)$ is maximized. Furthermore, if the policy is already confident and concentrated on other tokens, the collision probability $\sum_j P(j)^2$ remains large [9], sustaining high gradient norms.
- Gradient spikes due to $\frac{\pi_\theta(c|q,o)}{\pi_{\theta_{\text{old}}}(c|q,o)}$: When $\hat{A}_c < 0$, the ratio $\frac{\pi_\theta(c|q,o)}{\pi_{\theta_{\text{old}}}(c|q,o)}$ is only clipped from below (at $1 - \varepsilon$) and has no upper-bound. For a low-probability token, $\pi_{\theta_{\text{old}}}(c|\cdot)$ is extremely small. During PPO’s mini-batch updates, even a minor increase in $\pi_\theta(c|\cdot)$ can cause this ratio to become exceptionally large. When the advantage $\hat{A}_c$ is negative, this ratio is unclipped, leading directly to the gradient spikes observed in Fig. 3.

Misaligned Credit Assignment In addition to causing large gradients, the emergence of low-probability tokens undermines the credit assignment process in Zero RL. According to Fig. 4, low-probability tokens are more prone to accumulate with more turns, indicating higher token entropy and generation stochasticity. This compromises the overall success rate of responses with multi-turn TIR. In our A multi-turn trajectory that fails in its final turns receives a single negative reward for the entire sequence. This reward signal does not distinguish between the valid, high-probability reasoning tokens in early turns and the faulty, low-probability tokens that caused the eventual failure. This inherently discourages multi-turn behavior and collapses the policy toward single-turn, text-only reasoning.

3.3 Stabilizing Multi-Turn TIR Training with Void Turn Masking

Once we identify the emergence of low-probability tokens as the primary cause of multi-turn training, straightforward algorithmic improvements can be masking the policy loss on high-perplexity responses or truncating the importance ratio $\frac{\pi_\theta(c|q,o)}{\pi_{\theta_{\text{old}}}(c|q,o)}$. While these approaches prove effective in single-turn reasoning [10, 11, 12], we show in Fig. 5 (bottom) that they cannot resolve the training instability issue in multi-turn TIR. The threshold for loss masking or ratio truncating is hard to determine in a dynamic training process. Meanwhile, these approaches cannot efficiently filter out samples with incorrect credit assignment.

Nevertheless, loss masking on specific samples can indeed be helpful [13], as long as a proper filtering criterion is selected. From Fig. 4, we observe that the collapsed Turn 4 follows Turn 3 without a tool call. Intuitively, a turn with neither a tool call nor a final answer makes no contributions to the reasoning process and should not exist in an effective multi-turn trajectory. We define such turns as *void turns*, with Turn 3 in Fig. 4 as an example. In a response with low token probabilities and high generation stochasticity, a premature eos token is more likely to be generated, leading to the emergence of a void turn. As the void turn rarely exists in normal responses and can be the outcome of abnormal ones, we identify it as a key indicator of multi-turn instability.

This suggests void turns to be a simple rule of trajectory filtering, resulting in the SimpleTIR algorithm as demonstrated in Fig. 2. For each turn, we detect whether the response contains a complete code block or a final answer. If neither is present, we stop generation for that prompt and mask the policy loss of the entire response. This also corrects misaligned credit assignment by excluding normal early turns from being penalized. After trajectory filtering, we perform GRPO on effective multi-turn TIR trajectories. SimpleTIR is agnostic to specific RL methods and recent algorithmic modifications for LLM reasoning [10, 14, 15] are orthogonal to SimpleTIR’s trajectory filtering approach.

3.4 Implementation Details

In our training process, we adopt several practices to further improve efficiency and stability. For example, techniques from non-TIR reinforcement learning [16], such as clip higher, removing KL, dynamic sampling, and progressive length budget, are also effective in TIR training. Meanwhile,

Table 1: Performance comparison on various math benchmarks. Check and cross marks in the “TIR” column refers to whether the method involves TIR during training and evaluation. Slash, check, and cross marks in the “Zero RL” column refers to whether the model is untrained, trained with the Zero RL setting, or trained with other settings. The “From” column indicates the type of We fill the scores with - if they are not provided in respective reports.

Model	TIR	Zero RL	From	AIME24	AIME25	MATH500	Olympiad	AMC23	Hmmt 25
<i>Models based on Qwen2.5-7B</i>									
Qwen2.5-7B	✗	/	Base	3.2	1.1	51.9	15.4	21.7	0.0
Qwen2.5-7B-TIR	✓	/	Base	1.7	0.6	18.0	6.2	10.8	1.9
SimpleRL-Zoo-7B	✗	✓	Base	15.6	-	78.2	40.4	62.5	-
ToRL-7B	✓	✗	Math-Inst	40.2	27.9	82.2	49.9	75.0	-
Effective TIR-7B	✓	✗	Math	42.3	29.2	86.4	-	74.2	-
ARPO-7B	✓	✗	Inst	30.0	30.0	78.8	-	-	-
ZeroTIR-7B	✓	✓	Base	39.6	25.0	80.2	-	-	22.5
SimpleTIR-7B	✓	✓	Base	50.5	30.9	88.4	54.8	79.1	29.7
<i>Models based on Qwen2.5-32B</i>									
Qwen2.5-32B	✗	/	Base	4.2	1.6	43.1	17.8	28.0	0.2
Qwen2.5-32B-TIR	✓	/	Base	7.1	5.0	37.0	16.9	20.0	5.2
DAPO	✗	✓	Base	50.0	-	-	-	-	-
ReTool	✓	✗	Math-Inst	67.0	49.3	-	-	-	-
ZeroTIR-32B	✓	✓	Base	48	27	87.8	-	-	20.0
SimpleTIR-32B	✓	✓	Base	59.9	49.2	92.9	63.7	91.6	34.6

there are some implementation details specific to multi-turn TIR. First, we do not use chat templates, since special tokens like `|im_end|` are out-of-distribution for base models. Instead, when appending tool outputs, we simply prepend “Code Execution Result:” before the interpreter output. Second, we prepend every LLM-generated code block with a `final_answer` function, which allows the model in single-turn TIR to directly output the final answer inside a code block without an additional turn of querying. This technique provides a shortcut for simple tasks and improves training efficiency. Finally, we stop LLM generation after a complete code block and append the true tool outputs. We do not feed text tokens after each code block into the next turn. This technique helps prevent LLMs from hallucinating interpreter outputs after the code block.

4 Experiments

4.1 Setup

Training We prepare our training code with the VeRL [17] and Search-R1 [8] framework. We use Sandbox Fusion as an asynchronous code interpreter. The training datasets are Math3-5 from SimpleRL [18] and Deepscaler [19]. SimpleTIR follows the Zero RL setting and uses the unaligned Qwen-2.5 series as the base models, including Qwen-2.5-7B and Qwen-2.5-32B. During training, the rollout batch size is set to 512, and the mini update size is set to 128. The maximum response length is initially set to 16K, with a maximum of five turns of code execution. When the average response length plateaus, we increase the maximum response length to 24K and the largest number of turns to 10. Other training hyperparameters are in Appendix B.2.

Evaluation Our evaluation is conducted on Math500 [20], AIME24, AIME25, AMC23, and Hmmt Feb 25, using a temperature of 1 and reporting average@32 scores to reduce variance, following [16]. For comparison, we consider three categories of baselines. The first is non-TIR Zero RL, where we use SimpleRL-Zoo [18] and DAPO [16] as representative baselines. The performance gap between these methods and SimpleTIR highlights the advantage of incorporating TIR in mathematical reasoning. The second category is TIR RL from cold-start or specialized models, which includes ReTool [5], collecting cold-start datasets for supervised finetuning on Qwen2.5-Math-32B-Instruct, ARPO [21], finetuning Qwen2.5-7B-Instruct, as well as ToRL [22] and Effective CIR [23], both applying RL to the Qwen2.5-Math series. The final category is Zero RL with TIR, where, to the best of our knowledge, Zero-TIR [2] is the only method that strictly follows the Zero RL paradigm by training TIR models directly from base models.

Figure 5: Top: Training curves for SimpleTIR with different maximum number of turns. SimpleTIR with maximum 10 turns is resumed at 200 steps from SimpleTIR with maximum 5 turns. SimpleTIR clearly benefits from scaling interaction turns from 1 to 5. **Bottom:** The training curves for ablation studies in the first 320 steps. Trajectory filtering with high importance ratios or low probability tokens cannot resolve the challenge of training instability, while SimpleTIR suffers less from low probability tokens and gradient explosion.

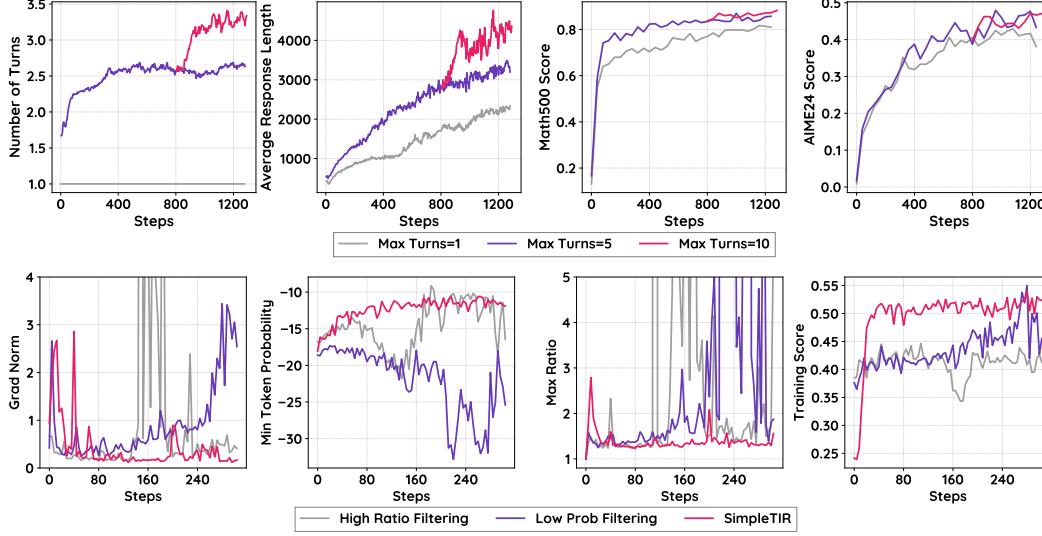


Table 2: Results of ablation studies. Considering the unstable training of ablated methods, we report the highest scores within 1000 gradient steps. “Naive Multi-Turn” directly applies RLVR in multi-turn TIR. “Low Prob” and “High Ratio” filtering refers to masking the policy loss on tokens with lowest probabilities or highest importance ratio.

	SimpleTIR-7B	Naive Multi-Turn	Low Prob Filtering	High Ratio Filtering	Stop Generation w/o Filtering
AIME24	50.5	20.8	23.3	26.3	26.1
Math500	88.4	73.1	72.8	75.0	77.3

4.2 Training Results

The training results are listed in Tab. 1. SimpleTIR demonstrates significant performance improvement over base models and outperform all baselines of Zero RL, either with or without TIR. SimpleTIR can also outperform baselines starting from Qwen2.5-Math-7B series, such as ToRL and Effective TIR. Comparing with methods not following Zero RL, it is shown that cold start significantly boosts performance, with ReTool-32B obtaining the highest scores on AIME24 and AIME25. The advantage of Zero RL over cold start lies in the diversity of reasoning patterns, as discussed in Sec. 4.4.

4.3 Training Curves and Ablation Studies

We show the training curve of SimpleTIR with 1, 5, and 10 turns of generation in Fig. 5 (Top). In all these settings, SimpleTIR exhibits constant and smooth increases of the average response length and performance scores. The average number of turns first arises quickly then remains constant for multi-turn SimpleTIR. We also observe that the response length and the Math500 score scales with more turns, while the AIME24 score does not benefit clearly. This indicates that different tasks require distinct reasoning patterns. Some may be solvable with few steps of reasoning, but others will take a number of external feedback before reaching the correct answer.

We also conduct ablation studies to demonstrate the effectiveness of trajectory filtering in SimpleTIR. We first investigate two alternative filtering criteria: high importance ratio and low token probabilities, as specified in the first paragraph of Sec. 3.3. As shown in Fig. 5 (Bottom), these two filtering

Figure 6: Demonstration of three reasoning patterns observed in responses generated by SimpleTIR.

Emergent Multi-Turn Reasoning Patterns

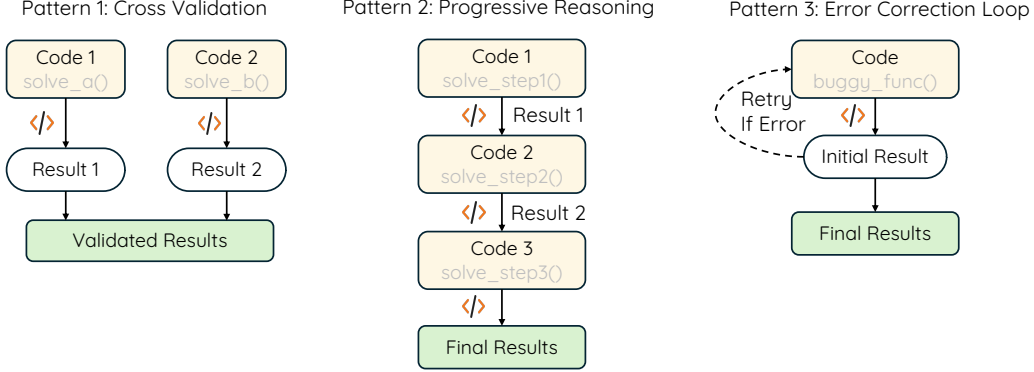


Table 3: Comparison of reasoning pattern frequencies in ReTool and SimpleTIR-32B responses. The summation of frequencies may exceed 100% as there may be more than one reasoning patterns in one response.

	Progressive Reasoning (%)	Cross Verification (%)	Error Correction (%)
ReTool	18.9	82.4	25.8
SimpleTIR-32B	46.5	86.0	38.0

approach cannot resolve the issue of gradient explosion, exhibiting unstable curves of training scores. SimpleTIR features a more stable curve of gradient norm, thanks to the mild token probability distributions. This demonstrates the effectiveness of void turn filtering in stabilizing multi-turn TIR training. We then consider an ablation method where LLM generation is terminated on void turns but resulting trajectories are not filtered when computing policy loss. According to the validation results in Tab. 2, this method is also inferior to SimpleTIR. This can be attributed to misaligned credit assignment since trajectories containing void turns can hardly obtain positive outcome. SimpleTIR handles such issue by masking the loss of whole responses containing void turns.

4.4 Emergence of Diverse Reasoning Behaviors

Thanks to the framework of Zero RL training, SimpleTIR automatically reinforces useful reasoning patterns obtained in the pretraining phase, rather than sticking to predefined patterns in the SFT dataset. In Appendix A.2, we show SimpleTIR responses with diverse multi-turn reasoning behaviors. They are mostly combinations of the three main reasoning patterns illustrated in Figure 6, namely Cross Validation, Progressive Reasoning, and Error Correction.

We also use Claude-3.7-Sonnet to identify and count the frequency of reasoning patterns in responses generated by ReTool and SimpleTIR-32B. The responses are filtered so that they all lead to the correct final answer. Both models demonstrate a strong tendency to conduct multiple rounds of cross verification. Meanwhile, SimpleTIR-32B exhibits more instances of progressive reasoning and error correction. This illustrates the advantage of Zero RL, which preserves more diversity in reasoning patterns.

5 Related Work

5.1 Zero RL for LLM Reasoning

DeepSeek-R1 [7] first shows that starting from an unaligned base model, large-scale RL training with outcome reward can unlock emergent chain-of-thought reasoning ability. Such paradigm is later referred to as Zero RL. SimpleRL [18] provides a reproducible cookbook to run Zero RL on

various open-source base models. Open-Reasoner-Zero [24] proposes that vanilla PPO with GAE ($\lambda = 1, \gamma = 1$) without KL regularization is sufficient to scale up Zero RL training. DAPO [16] introduce several training details that makes Zero RL training stable and efficient, such as raising the high clip ratio of PPO and GRPO and filtering tasks with 0 or 100% solve rate. Dr. GRPO [25] proposes to remove the length normalization term. SimpleTIR also follows the Zero RL pipeline and is orthogonal to training algorithms for Zero RL without TIR.

5.2 RL for Tool Integrated Reasoning

Several recent works focus on applying RL to improving the tool use ability of LLMs. Search-R1 [8] and R1-Search [26] focus on question-answering tasks, utilizing the search tool. For mathematical reasoning tasks, python interpreter can be a useful tool to conduct numerical calculations or enumerations. ReTool [5] employs a cold-start SFT phase before RL. ToRL [22] and Effective CIR [23] explore training recipes on math-specialized bases. These pipelines often rely on domain data, instruction tuning, or other supervision that introduce bias and complexity; in contrast, Zero RL is more general yet notoriously unstable in multi-turn settings. Our work directly addresses this stability gap under Zero RL by filtering trajectories with void turns. ZeroTIR [2] is also explicitly framed in the Zero RL setting. It proposes several stabilizing techniques that are orthogonal to our approach.

5.3 Stabilizing RL Training

Training instability is a significant challenge when applying RL to LLMs, often manifesting as entropy collapse and gradient norm explosions. Entropy-based methods explicitly maintain policy entropy or encourage re-generation at pivotal tokens to delay distributional narrowing [27, 28, 29]. Recent methods control the importance sampling ratio to reduce gradient variance and brittle updates by reweighting or constraining likelihood ratios, e.g., from token-level IS to sequence-level objectives and clipping [14, 15, 30, 31]. Data and trajectory filtering stabilizes training by discarding uninformative or harmful samples, e.g., multi-sample-then-filter schemes [32]. From the perspective of the learning signal itself, negative-only gradient updates have been shown to improve stability and generalization without sacrificing exploration, and more generally to focus updates on low-probability/high-entropy branching tokens [33]. SimpleTIR departs from the above methods by targeting the root cause specific to TIR, i.e., distribution shift induced by external tool outputs compounded by multi-turn error accumulation. It is also orthogonal to entropy regularization, IS ratio control, and negative-gradient schemes.

6 Conclusion

In this work, we introduce SimpleTIR, an RL framework designed to stabilize and enhance multi-turn TIR under the Zero RL setting. By addressing the key challenge of harmful negative samples via filtering out trajectories with void turns, our method achieves stable training dynamics and improves reasoning performance across a variety of mathematical benchmarks. Beyond state-of-the-art results, SimpleTIR also encourages the emergence of diverse reasoning patterns. These results highlight the potential of end-to-end multi-turn TIR RL, without relying on cold-start human data, as a pathway to scalable and reliable multi-turn reasoning in future LLM agent development.

Limitations and Future Work While effective, our method has several limitations. First, we use void turns as an indicator of low-probability tokens in multi-turn TIR. However, this indicator may not be directly applicable to tasks beyond multi-turn TIR. Second, we currently restrict the maximum number of turns to 10 for mathematical reasoning, though more interactions may be required for complex multi-turn agent tasks. Third, our training relies on a highly parallel sandbox for code execution. Therefore, the development of a faster and more reliable sandbox is an important direction for future work. Finally, achieving fully asynchronous rollout and reward calculation remains an open challenge. These limitations raise additional concerns around rollout efficiency, memory management, and credit assignment, which we leave for future exploration.

References

- [1] 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*, 2025.
- [2] Xinji Mai, Haotian Xu, Xing W, Weinong Wang, Yingying Zhang, and Wenqiang Zhang. Agent RL Scaling Law: Agent rl with spontaneous code execution for mathematical problem solving. *CoRR*, abs/2505.07773, 2025.
- [3] Carlo Baronio, Pietro Marsella, Ben Pan, and Silas Alberti. Multi-turn rl training for cuda kernel generation. <https://cognition.ai/blog/kevin-32b>, 2025.
- [4] Moonshot AI. Kimi-researcher: End-to-end rl training for emerging agentic capabilities. <https://moonshotai.github.io/Kimi-Researcher/>, June 2025.
- [5] 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. *CoRR*, abs/2504.11536, 2025.
- [6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
- [7] DeepSeek-AI Team. Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv:2501.12948*, 2025. URL <https://arxiv.org/abs/2501.12948>.
- [8] Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-RL: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025.
- [9] Yingru Li. Logit dynamics in softmax policy gradient methods. *CoRR*, abs/2506.12912, 2025.
- [10] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *CoRR*, abs/2507.18071, 2025.
- [11] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. Mastering complex control in MOBA games with deep reinforcement learning. In *AAAI*, pages 6672–6679. AAAI Press, 2020.
- [12] Yifan Zhang, Xingyu Lu, Xiao Hu, Chaoyou Fu, Bin Wen, Tianke Zhang, Changyi Liu, Kaiyu Jiang, Kaibing Chen, Kaiyu Tang, Haojie Ding, Jiankang Chen, Fan Yang, Zhang Zhang, Tingting Gao, and Liang Wang. RL-reward: Training multimodal reward model through stable reinforcement learning. *CoRR*, abs/2505.02835, 2025.
- [13] Wei Xiong, Jiarui Yao, Yuhui Xu, Bo Pang, Lei Wang, Doyen Sahoo, Junnan Li, Nan Jiang, Tong Zhang, Caiming Xiong, and Hanze Dong. A minimalist approach to LLM reasoning: from rejection sampling to reinforce. *CoRR*, abs/2504.11343, 2025.
- [14] Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training, August 2025. URL <https://fengyao.notion.site/off-policy-rl>.
- [15] 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.
- [16] Qiyang Yu, Zheng Zhang, Ruofei Zhu, et al. DAPO: An open-source llm reinforcement learning system at scale. *arXiv:2503.14476*, 2025. URL <https://arxiv.org/abs/2503.14476>.
- [17] Guangming Sheng, Chi Zhang, Zilinfeng 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.

- [18] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerrl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *CoRR*, abs/2503.18892, 2025.
- [19] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. <https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>, 2025. Notion Blog.
- [20] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*, 2021.
- [21] Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: end-to-end policy optimization for GUI agents with experience replay. *CoRR*, abs/2505.16282, 2025.
- [22] Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling tool-integrated RL. *CoRR*, abs/2503.23383, 2025.
- [23] Fei Bai, Yingqian Min, Beichen Zhang, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. Towards effective code-integrated reasoning. *arXiv preprint arXiv:2505.24480*, 2025.
- [24] Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv:2503.24290*, 2025. URL <https://arxiv.org/abs/2503.24290>.
- [25] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding rl-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.
- [26] 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. *CoRR*, abs/2503.05592, 2025.
- [27] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.
- [28] Qingbin Li, Rongkun Xue, Jie Wang, Ming Zhou, Zhi Li, Xiaofeng Ji, Yongqi Wang, Miao Liu, Zheming Yang, Minghui Qiu, et al. Cure: Critical-token-guided re-concatenation for entropy-collapse prevention. *arXiv preprint arXiv:2508.11016*, 2025.
- [29] Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*, 2025.
- [30] Yuzhong Zhao, Yue Liu, Junpeng Liu, Jingye Chen, Xun Wu, Yaru Hao, Tengchao Lv, Shaohan Huang, Lei Cui, Qixiang Ye, et al. Geometric-mean policy optimization. *arXiv preprint arXiv:2507.20673*, 2025.
- [31] 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.
- [32] Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning. *arXiv preprint arXiv:2508.09726*, 2025.
- [33] Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in llm reasoning. *arXiv preprint arXiv:2506.01347*, 2025.

A Example Responses

A.1 Incomplete Response

We present representative failure cases that contain void turns, i.e., turns that produce neither a complete, executable code block nor a boxed final answer. These examples serve a diagnostic role: they illustrate how OOD tool feedback and compounding errors precipitate collapsed generations and gradient spikes during Zero RL. Tab. 4 shows a typical trajectory in which a void turn disrupts subsequent decoding and leads to corrupted outputs, motivating our trajectory filtering rule.

A.2 Response with Emergent Reasoning Behaviors

We provide qualitative rollouts that demonstrate the diverse multi-turn behaviors SimpleTIR elicits without instruction-level biases. Tab 5 illustrates progressive reasoning with code improvement. Taken together with the quantitative pattern analysis in the main text, these cases substantiate our claim that Zero RL with TIR encourages richer strategies than cold-start SFT.

B Experiments

B.1 Prompt for Multi-turn TIR Generation

We include the exact prompt template used to generate multi-turn TIR trajectories in Tab. 6b. The design emphasizes: (1) selective use of Python wrapped in triple backticks as complete scripts (with imports); (2) explicit printing of intermediate quantities so that execution feedback can guide later turns; and (3) a standardized answer channel (`final_answer(...)` or `\boxed{...}`) that cleanly terminates trajectories when a solution is reached. These choices stabilize interaction with the interpreter, reduce format variance, and make it easy to detect valid tool calls versus void turns.

B.2 Hyperparameters

We show the training hyperparameters of SimpleTIR in Tab. 6a. Below we explain the rationale behind the hyperparameters. We cap the initial max response length at 16,384 tokens to accommodate complete code blocks and verbose execution traces without premature truncation. Initial max interaction turns = 5 bounds episode length and compute while still allowing the model to plan, execute, and verify within a single trajectory. We set rollout temperature = 1.0 to preserve diversity in candidate solutions and rely on selection/credit assignment rather than explicit entropy bonuses to drive exploration. Each update uses a sampling batch size of 1,280 responses with $n = 16$ rollouts per prompt, which yields broad coverage of tool-use strategies per input while keeping variance manageable.

We use standard PPO with clip ratio = 0.2 / 0.28 (low/high) to constrain policy updates; the slightly looser upper bound avoids over-penalizing advantageous moves identified by execution feedback. PPO epochs = 4 provide sufficient reuse of samples without overfitting to batch noise. The train batch size = 512 balances gradient estimate quality and memory use. A small actor learning rate = $1e-6$ and gradient clipping = 1 (global-norm) prevent instability from long-context backprop and occasional high-magnitude advantages produced by sparse rewards. We treat each dialogue as an undiscounted episodic task with $\gamma = 1.0$. We also set entropy coefficient = 0 and KL coefficient = 0. Exploration is instead induced by temperature and multi-rollout sampling.

B.3 Prompt for Extracting Reasoning Patterns

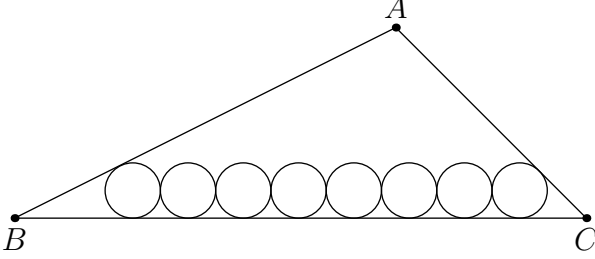
To analyze behaviors systematically, in Tab. 7 we provide the prompt used to label reasoning behaviors in trajectories. It first enforces minimal bookkeeping (e.g., number of code blocks, whether any execution equals the final answer), then queries four patterns: decomposed sub-solves (with non-final execution), induction, self-verification, and bug-fixing via additional code. The constrained output format improves label consistency and allows fair frequency comparisons across methods.

Table 4: An example from SimpleTIR-7B on AIME24 dataset. Responses generated in different turns are separated with a vertical line. Response Part-II contains a void turn, where no complete code blocks nor boxed answers are generated. Both in and after the void turn, the model generates corrupted responses.

Example #1 from AIME24 Dataset

Question:

Eight circles of radius 34 are sequentially tangent, and two of the circles are tangent to AB and BC of triangle ABC , respectively. 2024 circles of radius 1 can be arranged in the same manner. The inradius of triangle ABC can be expressed as $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m + n$.



Labeled Answer:

197

Model Output by SimpleTIR-7B (Part-I)

To solve this problem, we need to determine the inradius of triangle ABC given the arrangement of circles. Here is the step-by-step reasoning:

****Understand the Geometry:****

- We have eight circles of radius 34 that are tangent to each other sequentially, with two of them being tangent to the sides AB and BC of triangle ABC . - Additionally, 2024 circles of radius 1 can fit in the same manner.

****Determine the Distance Between Centers of the Larger Circles:****

- The centers of the eight tangent circles of radius 34 will form a chain where each circle is tangent to the next, meaning the distance between the centers of two consecutive circles is $34 + 34 = 68$. - Since there are 8 circles, the total distance from the center of the first circle to the center of the last circle would be $68 \times (8 - 1) = 68 \times 7 = 476$. However, we need to consider that the first circle is tangent to AB and the last circle is tangent to BC . This distance of 476 actually represents the length of the tangent chain along the line connecting the points of tangency on AB and BC .

****Relate to Triangle ABC :**

- If we look at the centers of these circles, they form a line parallel to BC and AB , and the distance between this line and either AB or BC is equal to the radius of the circles (34 for the larger circles, 1 for the smaller circles).

****Calculate the Lengths:****

- For the larger circles (radius 34):

- The total distance connecting the points of tangency on AB and BC is effectively the distance between the parallel line formed by the centers of these circles and the line BC or AB , which is $8 \times 34 = 272$ (since there are 8 circles, and each has a radius of 34, the total vertical distance covered would be the diameter of 7 circles if we think about it in terms of centers plus radius). However, since they are tangent to each other sequentially, the distance between the lines AB and BC (parallel to the line connecting the centers) should be $34 + 34 \times (8 - 1) = 34 + 34 \times 7 = 34 \times 8 = 272$.

- For the smaller circles (radius 1):

- Similarly, if we can fit 2024 circles of radius 1, the total vertical distance from the line of tangency on AB to the line of tangency on BC would be $2024 \times 1 = 2024$.

****Inradius Relationship:****

- The inradius r of triangle ABC is the radius of the inscribed circle. To find this, we need to understand how the heights and lengths of the triangle are affected by these tangent circles.

- Let's denote the length of $AB = c$, $BC = a$, and $CA = b$. The height from A to BC can be considered by looking at how the circles stack up vertically.

- For the larger circles:

- If two circles are tangent to AB and BC , the length along AB and BC would create a configuration where if you think about the points of tangency, the distance covered by the centers vertically for 8 circles should match up with the triangle height minus twice the radius (since the circles are also on the boundary of the triangle).

Model Output by SimpleTIR-7B (Part-II)

.....Some Response Omitted.....

Given this, distance between sides should follow proportionally: Distance apart: smaller = $\frac{2024 \times 1}{8 \times 34}$ Total height h of the triangle should include this detailed intermediary:

Let's re-evaluate and verify it step-by-step via coding:

```
import sympy as sp
# Define the radius of larger and smaller circles
R_large = 34
R_small = 1
num_large_circles = 8
num_small_circles = 2024
# Total vertical distance covered by the larger circles' centers
# from base to the point of tangency
distance_large_centers = num_large_circles * (2 * R_large)
# How many points can fit in this same height but smaller the
# vertical line can stack:
distance_small_centers = num_small_circles * (2 * R_small)
# Height from tangency to the actual top parallel line keeping them
# tangent:
height_large = distance_large_centers # actually includes the
# tangency start:
height_small = distance_small_centers
# Deriving the total height by proportionality:
R_ratio = R_small / R_large
num_ratio = num_small_circles / num_large_circles
HR_length = (num_large_circles * 2 * R_large) # covering space.
H_effective = HR_length
# This total covers height only parallelly: from base and top
# distance:
h_total = HR_length
inradius = height_large / (num_ratio + 1)
inradius = sp.Rational(h_total / (num_ratio + 1))
print(f"calculated inradius: {inradius}")
```

Code execution result: calculated inradius: 4822752356869193/2251799813685248

This calculation seems overly complex; let's simplify using simpler, geometric visualization and relationships: Simplified Geometric Relationship:

****Large Circle Tangent Arrangement:****

- We have 8 circles of radius 34, and 2 of these circles are tangent to AB and BC . The effective vertical distance taken by these circles, considering their arrangement repeating pattern of centers, is given by:

$$8 \cdot (2 \cdot 34) = 8 \cdot 68$$

****Small Circle Arrangement:****

- For smaller circles of radius 1, such that 2024 circles fit similarly: - The vertical distance arrangement for 2024 circles would be:

$$2024 \cdot 2 \cdot 1 = 2024 \cdot 2 = 4048$$

.....Some Response Omitted.....

We repeat arranging parallel: Consider:

each pair arrangement, verifying circles top-height Even scan: If each arrangement euclidean, height method involving polygons: solved for ratio translating through unique noting: Combining parameters: This par; valid:

$\lfloor \frac{m}{n} \rfloor$ sum ratio sorts in idlegate ration 1}. However, i' sequential seek refining this calculation more consistently: iterating setup considered overall expression precise: Now we solve ensuring configurations understandable suggest circle distributions proportionally: If):

- $\lfloor \frac{m}{n} \rfloor$ each step repeated configurations: compute $\lfloor \frac{m}{n} \rfloor$ largest Thus: We need to find side cognize area further on: triangle holds hold altitude relation contractive step:

Table 5: An example from SimpleTIR-7B on AIME24 dataset. It demonstrates the behavior of progressive reasoning and code improvement. Responses generated in different turns are separated with a vertical line.

Example #1 from AIME24 Dataset
Question: Let $ABCD$ be a tetrahedron such that $AB = CD = \sqrt{41}$, $AC = BD = \sqrt{80}$, and $BC = AD = \sqrt{89}$. There exists a point I inside the tetrahedron such that the distances from I to each of the faces of the tetrahedron are all equal. This distance can be written in the form $\frac{m\sqrt{n}}{p}$, where m, n, and p are positive integers, m and p are relatively prime, and n is not divisible by the square of any prime. Find $m + n + p$. Labeled Answer: 104
Model Output by SimpleTIR-7B (Part-I)
To solve the given problem, we need to find the distance from point I (which is the incenter of the tetrahedron $ABCD$) to each of the faces of the tetrahedron, given that the distances are all equal. This distance is the radius r of the inscribed sphere of the tetrahedron. First, we need to determine the volume V of the tetrahedron and the surface area S. The formula for the radius r of the inscribed sphere is given by: $r = \frac{3V}{S}$ ### Step 1: Calculate the Volume V of the Tetrahedron We can use the Cayley-Menger determinant to find the volume of the tetrahedron. The Cayley-Menger determinant for a tetrahedron with edge lengths a, b, c, d, e, f (where $a = AB, b = AC, c = AD, d = BC, e = BD, f = CD$) is given by: $288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & a^2 & b^2 & c^2 & \\ 1 & a^2 & 0 & d^2 & e^2 & \\ 1 & b^2 & d^2 & 0 & f^2 & \\ 1 & c^2 & e^2 & f^2 & 0 & \end{vmatrix}$ Substituting $a = \sqrt{41}, b = \sqrt{80}, c = \sqrt{89}, d = \sqrt{89}, e = \sqrt{80}, f = \sqrt{41}$: $288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 41 & 80 & 89 & \\ 1 & 41 & 0 & 89 & 80 & \\ 1 & 80 & 89 & 0 & 41 & \\ 1 & 89 & 80 & 41 & 0 & \end{vmatrix}$ We can calculate the determinant of the matrix: <pre> import sympy as sp # Define the matrix matrix = sp.Matrix([[0, 1, 1, 1, 1], [1, 0, 41, 80, 89], [1, 41, 0, 89, 80], [1, 80, 89, 0, 41], [1, 89, 80, 41, 0]]) # Calculate the determinant det = matrix.det() # Volume squared V_squared = det / 288 # Volume V = sp.sqrt(V_squared) print(f"Volume V: {V}") </pre> Code Execution Result: Volume V: 160/3

Model Output by SimpleTIR-7B (Part-II)

Step 2: Calculate the Surface Area S of the Tetrahedron

The surface area S is the sum of the areas of the four triangular faces. We can find the area of each triangle using Heron's formula.

For triangle ABC :

$$- AB = \sqrt{41} - AC = \sqrt{80} - BC = \sqrt{89}$$

The semi-perimeter s is:

$$s = \frac{\sqrt{41} + \sqrt{80} + \sqrt{89}}{2}$$

Using Heron's formula:

$$\text{Area} = \sqrt{s(s - AB)(s - AC)(s - BC)}$$

We need to find the areas of triangles ABD , ACD , and BCD similarly. Let's calculate these areas step by step using Python.

```
import sympy as sp

# Define the edge lengths
AB = sp.sqrt(41)
AC = sp.sqrt(80)
AD = sp.sqrt(89)
BC = sp.sqrt(89)
BD = sp.sqrt(80)
CD = sp.sqrt(41)

# Function to compute the area of a triangle using Heron's formula
def triangle_area(a, b, c):
    s = (a + b + c) / 2
    area = sp.sqrt(s * (s - a) * (s - b) * (s - c))
    return area

# Calculate the areas of the triangular faces
area_ABC = triangle_area(AB, AC, BC)
area_ABD = triangle_area(AB, AD, BD)
area_ACD = triangle_area(AC, AD, CD)
area_BCD = triangle_area(BC, BD, CD)

# Total surface area
S = area_ABC + area_ABD + area_ACD + area_BCD

print(f"Area of triangle ABC: {area_ABC}")
print(f"Area of triangle ABD: {area_ABD}")
print(f"Area of triangle ACD: {area_ACD}")
print(f"Area of triangle BCD: {area_BCD}")
print(f"Total surface area S: {S}")
```

Code Execution Result: Area of triangle ABC: $\sqrt{(-\sqrt{41})/2 + 2\sqrt{5} + \sqrt{89})/2} \sqrt{(\sqrt{41})/2 + 2\sqrt{5} + \sqrt{89})/2} \sqrt{(-\sqrt{89})/2 + \sqrt{41})/2 + 2\sqrt{5}}$
Area of triangle ABD: $\sqrt{(-\sqrt{41})/2 + 2\sqrt{5} + \sqrt{89})/2}$
...This content has been truncated to stay below 512 characters...
) $\sqrt{(-\sqrt{89})/2 + \sqrt{41})/2 + 2\sqrt{5}}$ Total surface area S: $4\sqrt{(-\sqrt{41})/2 + 2\sqrt{5} + \sqrt{89})/2} \sqrt{(\sqrt{41})/2 + 2\sqrt{5} + \sqrt{89})/2} \sqrt{(-\sqrt{89})/2 + \sqrt{41})/2 + 2\sqrt{5}}$

It appears the individual areas calculated by Heron's formula are quite complicated and expressed in a form needing simplification to find the total surface area S . Let's simplify the expressions to make the calculations clearer and ensure we obtain a numerical value for the surface area S . We can use simplify to get more manageable expressions and then compute the numerical value. Let's also calculate $r = \frac{3V}{S}$ step by step to simplify finding the equal distances. We know the volume $V = \frac{160}{3}$.

Model Output by SimpleTIR-7B (Part-III)

We will now simplify our approach and focus on calculating these areas numerically to find S . Afterward, we will compute r and ensure the result is in the required form $\frac{m\sqrt{n}}{p}$. Let's simplify our strategy and calculate this accurately in Python.

```
import sympy as sp
# Define symbolic variables
sqrt_41 = sp.sqrt(41)
sqrt_80 = sp.sqrt(80)
sqrt_89 = sp.sqrt(89)
# Function to compute area using Heron's formula
def area_heron(a, b, c):
    s = (a + b + c) / 2
    area = sp.sqrt(s * (s - a) * (s - b) * (s - c))
    return sp.simplify(area)
# Calculate areas of the triangles
area_ABC = area_heron(sqrt_41, sqrt_80, sqrt_89)
area_ABD = area_heron(sqrt_41, sqrt_89, sqrt_80)
area_ACD = area_heron(sqrt_80, sqrt_89, sqrt_41)
area_BCD = area_heron(sqrt_89, sqrt_80, sqrt_41)
# Total surface area
S = area_ABC + area_ABD + area_ACD + area_BCD
# Volume
V = sp.Rational(160, 3)
# Radius of the inscribed sphere
r = 3 * V / S
# Simplify r
r_simplified = sp.simplify(r)
print("Radius r:", r_simplified)
```

Code Execution Result: Radius r : $20\sqrt{21}/63$

The radius r of the inscribed sphere is $\frac{20\sqrt{21}}{63}$.

This is in the form $\frac{m\sqrt{n}}{p}$, where $m = 20$, $n = 21$, and $p = 63$. Here, m and p are relatively prime, and n is not divisible by the square of any prime.

Step 3: Finding $m + n + p$

Now, we need to calculate $m + n + p$:

$$m + n + p = 20 + 21 + 63$$

Let's calculate this:

```
import sympy as sp

# Values
m = 20
n = 21
p = 63

# Sum
result = m + n + p
print(result)
```

Code Execution Result: 104

The value of $m + n + p$ is $\boxed{104}$.

This completes the solution. The distance from point I to each of the faces of the tetrahedron is $\frac{20\sqrt{21}}{63}$, and the final answer is $\boxed{104}$.

Table 6: Key hyperparameters (left) and the prompt for generating multi-turn TIR responses (right).

(a) Key Hyperparameters		(b) The Prompt for Generating Multi-turn TIR Responses
Hyperparameter	Value	Solve the following problem step by step. You now have the ability to selectively write executable Python code to enhance your reasoning process. The Python code will be executed by an external sandbox, and the output (after “Code execution result: ”) is returned to aid your reasoning and help you arrive at the final answer. The Python code should be complete scripts, including necessary imports. Code Format: Each code snippet is wrapped between ```. You need to use <code>print()</code> to output intermediate results. Answer Format: You can use the <code>final_answer()</code> function in the code to return your final answer. For example, to answer the User Question: What is the result of the $5 + 3 + 1294.678$?, you can write: <pre>answer = 5 + 3 + 1294.678 final_answer(answer)</pre> You can also use <code>\boxed</code> to return your answer. The last part of your response should be: <code>\boxed{“The final answer goes here.”}</code> User Question:
Initial max response length	16384	
Rollout Temperature	1	
Initial max interaction turns	5	
Train batch size	512	
Sampling batch size	1280	
Rollouts per prompt (n)	16	
PPO clip ratio (low / high)	0.2 / 0.28	
Entropy coefficient	0	
Discount factor γ	1.0	
GAE λ	1.0	
KL coefficient (β)	0	
PPO epochs	4	
Actor learning rate	1e-6	
Gradient Clipping	1	

Table 7: The prompt that instructs Claude-3.7-Sonnet to extract reasoning patterns from the TIR trajectories.

I have a reasoning process of an LLM. The LLM can write code and get code execution result. According to the following reasoning process, please first answer the following questions:

1. Is the code execution result or interpreter output equal to the final answer?
2. How many code blocks are there in the reasoning process?
3. If there are several code blocks, are the code execution results all the same?

Format:

1. xxx
2. xxx
3. xxx

Please then determine whether the following reasoning process contains following four reasoning patterns:

1. Include at least two code blocks, each solving unique sub-questions. ****Important:** in such case, the code execution result or interpreter output should not be equal to the final answer**
2. Use induction, from special case to general conclusions
3. Use code or text to do self-verification
4. Write another code block when the previous code has some bugs

Format:

Reasoning Pattern 1: Yes/No
Reasoning Pattern 2: Yes/No
Reasoning Pattern 3: Yes/No
Reasoning Pattern 4: Yes/No

Please do not output any other words.

Reasoning process: