

AdaTIR: Adaptive Tool-Integrated Reasoning via Difficulty-Aware Policy Optimization

Zhaiyu Fang*, Ruipeng Sun
Trip.com Group
{zhaiyufang, sunrp}@trip.com

Abstract

Tool-Integrated Reasoning (TIR) improves the problem-solving ability of Large Language Models (LLMs), but current TIR agents can invoke external tools even when internal reasoning is sufficient. This cognitive offloading increases latency and can expose the model to noisy tool feedback. We propose AdaTIR, a difficulty-aware policy optimization framework that applies efficiency pressure when a task appears solvable under the current TIR policy. This encourages less tool-dependent reasoning on easier tasks while preserving tool access for harder ones. We further identify a sign-reversal failure in group-normalized RL: once rollouts in a group are mostly correct, an auxiliary tool-use penalty can dominate the correctness signal and assign negative advantages to correct but tool-heavy trajectories. AdaTIR addresses this with Clipped Advantage Shaping (CAS), which keeps efficiency as a bounded secondary signal. Across mathematical benchmarks, AdaTIR reduces average tool calls by up to 97.6% while preserving a competitive accuracy-efficiency trade-off, and improves AIME 2024 accuracy by 4.8% when tool access is disabled. Additional results on search-based QA and mixed-tool settings provide preliminary, directional evidence beyond Python-based math reasoning.

1 Introduction

Large Language Models (LLMs), including OpenAI-o1 (OpenAI et al., 2024) and DeepSeek-R1 (DeepSeek-AI et al., 2025), have shown strong reasoning ability on mathematical problem solving (Shao et al., 2024), repository-level code generation (Jimenez et al., 2024), and open-domain question answering (Kwiatkowski et al., 2019). **Tool-Integrated Reasoning (TIR)** extends this ability by allowing models to interleave natural-language reasoning with external executors (Gou et al., 2024;

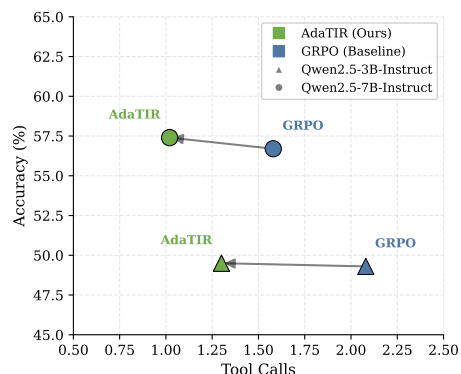


Figure 1: **Accuracy versus Tool Calls for Qwen2.5 under different training strategies.** GRPO and AdaTIR denote the baseline and our proposed method, respectively. AdaTIR reduces tool use while keeping comparable accuracy on both 3B and 7B backbones.

Li et al., 2025c; Feng et al., 2025; Singh et al., 2025; Xue et al., 2025). This pattern is increasingly relevant to NLP agents that combine language reasoning with search, code execution, or structured external services. Yet tool access also introduces a failure mode: models may offload simple intermediate steps to tools by default (Wang et al., 2025a; Xu et al., 2024). This raises latency and can derail reasoning when tool outputs are noisy or irrelevant. The central problem is not whether tools are useful, but when tool use should be suppressed without damaging performance on genuinely tool-dependent tasks.

Existing efficiency methods only partially address this problem. Inference-time budgets (Liu et al., 2025) constrain behavior after training and may cause dead ends when the model has not internalized the required reasoning (Liu et al., 2024; Shinn et al., 2023). Training-time penalties such as OTC reduce tool calls, but an indiscriminate penalty creates a direct trade-off: a strong penalty removes redundant calls on easy tasks, while the same pressure can suppress necessary calls on hard

* Corresponding author.

tasks. AdaTIR instead treats efficiency as difficulty-dependent. It applies tool-use pressure primarily to tasks that the current policy already solves reliably, using group success rate as a zero-extra-compute proxy for online difficulty.

We propose **AdaTIR**, a difficulty-aware policy optimization framework for adaptive tool use. The goal is not global tool aversion. AdaTIR encourages less tool-dependent reasoning when tool use is likely redundant, while keeping tool access available when it is needed for correctness. This design uses the model’s current success rate as a lightweight signal for deciding when efficiency pressure is less likely to conflict with task success.

This strategy exposes a second optimization issue. Under group-normalized RL, once most roll-outs in a group are correct, correctness no longer provides a strong relative signal. A naive efficiency penalty can then dominate the advantage estimate, giving a negative update to a correct trajectory simply because it used more tools than its peers. We call this sign reversal. AdaTIR uses **Clipped Advantage Shaping (CAS)** to reduce this failure mode by adding efficiency as a bounded auxiliary advantage rather than folding it directly into the scalar reward. CAS helps keep correctness as the primary learning signal while limiting large efficiency-driven updates (Wang et al., 2025c; Cui et al., 2025).

Our contributions are summarized as follows:

- We propose AdaTIR, a framework that uses online difficulty estimates to regulate tool invocations and reduce redundant cognitive offloading on easier tasks.
- We introduce Clipped Advantage Shaping (CAS), which bounds the efficiency signal relative to correctness and reduces large negative updates on correct but tool-heavy trajectories under group-normalized RL.
- We report accuracy-efficiency results across math, search-based QA, and mixed-tool settings. On the main math benchmarks, AdaTIR reduces tool calls by up to 97.6% while preserving competitive accuracy, and improves AIME 2024 accuracy in a no-tool evaluation setting.

2 Preliminaries

Tool-Integrated Reasoning (TIR). In the TIR paradigm, the large language model acts as a rea-

soning engine that interleaves text generation with external tool interactions (Wang et al., 2023; Shao et al., 2024). Given a task q , the reasoning process of the policy π_θ forms a trajectory τ consisting of sequential reasoning steps and environment feedbacks:

$$\tau = (w_1, c_1, o_1, w_2, c_2, o_2, \dots, w_K, c_K, o_K, \hat{y}) \quad (1)$$

where w_k represents the k -th reasoning thought, c_k is the tool invocation command, and o_k is the corresponding observation returned by code execution.

Group Relative Policy Optimization (GRPO). We adopt GRPO as our base optimization framework. In contrast to traditional Actor-Critic (Konda and Tsitsiklis, 1999) methods that require a value function, GRPO estimates the baseline directly from group scores, reducing memory overhead. The training objective is defined as:

$$\mathcal{J}_{\text{GRPO}} = \frac{1}{G} \sum_{i=1}^G \frac{1}{|\tau_i|} \sum_{t=1}^{|\tau_i|} \left(\min(r_{i,t} \hat{A}_{i,t}, \text{clip}(r_{i,t}, 1 - \epsilon, 1 + \epsilon) \hat{A}_{i,t}) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right) \quad (2)$$

where the importance sampling ratio $r_{i,t}$ is explicitly given by:

$$r_{i,t} = \frac{\pi_\theta(\tau_{i,t} \mid q, \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,t} \mid q, \tau_{i,<t})} \quad (3)$$

Here, $\tau_{i,t}$ denotes the t -th token in trajectory τ_i , and $\hat{A}_{i,t}$ represents the normalized advantage computed within group i .

3 Method

AdaTIR optimizes tool use by separating two questions: whether a task appears solvable under the current TIR policy, and whether a correct trajectory used more tools than necessary. Figure 2 summarizes the framework. We first define a difficulty-aware efficiency reward that targets redundant tool use on low-difficulty tasks (§3.1). We then introduce Clipped Advantage Shaping (CAS), which keeps this efficiency signal subordinate to correctness during policy optimization (§3.2).

3.1 Difficulty-Aware Efficiency Reward for Adaptive Tool Use

The difficulty-aware reward is designed to penalize redundant tool calls only when doing so is unlikely to harm correctness. Unlike methods that apply

penalty to successful trajectories that rely on redundant tool use. We first define a sine-based penalty factor $f(\tau_i)$ to quantify the tool invocation cost:

$$f(\tau_i) = \sin\left(\frac{\pi}{2} \cdot \frac{N(\tau_i)}{N_{\max}}\right) \quad (5)$$

where $N(\tau_i)$ is the number of tool calls, and $N_{\max}$ represents the tool call limit. To prioritize correctness and prevent reward hacking, this penalty is conditionally applied only to correct trajectories ($r_{\text{acc},i} = 1$) on simple tasks ($\phi_q < \phi_{\text{low}}$):

$$r_{\text{eff},i} = \begin{cases} -\alpha \cdot f(\tau_i), & \text{if } r_{\text{acc},i} = 1 \\ & \text{and } \phi_q < \phi_{\text{low}} \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

where α is the efficiency-penalty scale and ϕ_{low} is the difficulty threshold. The final reshaped reward for the i -th trajectory is:

$$r_{\text{total},i} = r_{\text{acc},i} + r_{\text{eff},i} \quad (7)$$

By applying the efficiency penalty only to correct trajectories on low-difficulty tasks, AdaTIR discourages unnecessary tool calls without rewarding incorrect no-tool answers. Appendix F.1 discusses the penalty shape.

3.2 Clipped Advantage Shaping for Stable Training

Training Instability of Reward Shaping. Directly incorporating $r_{\text{eff},i}$ into the GRPO objective can induce training instability (Lin and Xu, 2025). Substituting the total reward $r_{\text{total},i}$ into the standard advantage formulation yields:

$$A_i = \frac{(r_{\text{acc},i} + r_{\text{eff},i}) - \text{mean}(\mathbf{r}_{\text{acc}} + \mathbf{r}_{\text{eff}})}{\text{std}(\mathbf{r}_{\text{acc}} + \mathbf{r}_{\text{eff}})} \quad (8)$$

When a task is simple or the model reaches performance saturation, all rollouts within a group could be correct ($r_{\text{acc},i} = 1$). In such scenarios, the accuracy signal becomes negligible in the numerator, and the advantage estimation degenerates into a form driven exclusively by efficiency:

$$A_i \approx \frac{r_{\text{eff},i} - \text{mean}(\mathbf{r}_{\text{eff}})}{\text{std}(\mathbf{r}_{\text{eff}})} \quad (9)$$

This can assign negative advantages to correct trajectories with above-average tool invocation. Furthermore, if the variance of $\mathbf{r}_{\text{eff}}$ is small, the advantage estimate becomes sensitive to minor differences in tool invocation. This produces noisy

learning signals and may trigger premature mode collapse.

Clipped Advantage Shaping (CAS). To mitigate this instability, CAS decouples the estimation of correctness and efficiency. Specifically, the accuracy advantage A_i^{acc} and the raw efficiency advantage A_i^{eff} are computed by applying group normalization to $r_{\text{acc},i}$ and $r_{\text{eff},i}$, respectively. The CAS advantage is formulated as:

$$A_i^{\text{CAS}} = A_i^{\text{acc}} + r_{\text{acc},i} \cdot \gamma \cdot A_i^{\text{eff, clip}} \quad (10)$$

where the binary mask $r_{\text{acc},i} \in \{0, 1\}$ restricts efficiency updates to correct trajectories, and $\gamma > 0$ scales the efficiency incentive. The clipped efficiency signal is defined as:

$$A_i^{\text{eff, clip}} = \text{clip}(A_i^{\text{eff}}, -\delta|A_i^{\text{acc}}| - \eta, \delta|A_i^{\text{acc}}| + \eta) \quad (11)$$

where $\delta \in [0, 1)$ and $\eta > 0$ control the clipping range relative to the correctness signal.

This formulation keeps the efficiency incentive bounded relative to A_i^{acc} . It reduces the chance that a correct trajectory receives a large negative update only because it used more tools than other correct trajectories. The clipping mechanism also limits advantage inflation, keeping auxiliary feedback more stable across task groups. Appendix F.2 gives a condition-level analysis of the CAS update.

4 Experiments

4.1 Experimental Setup

Training We implement AdaTIR on Qwen2.5-3B/7B-Instruct backbones (Qwen et al., 2025) using the veRL framework (Sheng et al., 2025) and SandboxFusion (Bytedance-Seed-Foundation-Code-Team et al., 2025). We utilize ReTool-SFT¹ for the SFT stage and DAPO-Math-17k² for RL. Following Feng et al. (2025), we set the maximum sequence and response lengths to 4096. During RL, the tool-call limit is $N_{\max} = 4$. AdaTIR uses $\phi_{\text{low}} = 0.8$, CAS scale $\gamma = 0.9$, clipping ratio $\delta = 0.99$, and clipping offset $\eta = 10^{-4}$; all other hyperparameters follow the ReTool recipe (Appendix A).

Evaluation We evaluate AdaTIR on AIME 2024/2025, AMC 23, and GSM8K (Cobbe et al.,

¹<https://huggingface.co/datasets/JoeYing/ReTool-SFT>

²<https://huggingface.co/datasets/BytedTsinghua-SIA/DAPO-Math-17k>

Method	AIME 2024		AIME 2025		AMC23		GSM8K	
	Acc ($\uparrow$)	ATC ($\downarrow$)	Acc ($\uparrow$)	ATC ($\downarrow$)	Acc ($\uparrow$)	ATC ($\downarrow$)	Acc ($\uparrow$)	ATC ($\downarrow$)
Qwen2.5-3B-Instruct	10.0	–	6.7	–	38.0	–	50.0	–
Qwen2.5-3B-Instruct-TIR	6.0	0.79	1.3	0.71	30.0	1.02	36.0	1.58
Qwen2.5-3B-Instruct (GRPO)	26.3	2.52	24.2	2.58	60.2	2.02	86.5	1.19
AdaTIR-3B (Ours)	29.2	1.63	23.8	2.01	58.0	1.35	87.1	0.22
Qwen2.5-7B-Instruct	12.3	–	10.0	–	52.0	–	61.9	–
Qwen2.5-7B-Instruct-TIR	5.0	0.48	5.4	0.36	21.3	0.51	17.3	0.51
Qwen2.5-7B-Instruct (GRPO)	33.8	2.02	27.1	2.01	74.7	1.44	91.0	0.83
AdaTIR-7B (Ours)	37.1	1.45	27.3	1.57	72.3	1.02	92.8	0.02

Table 1: Performance Comparison on Various Mathematical Reasoning Benchmarks. We report Avg@16 Accuracy and Average Tool Calls (ATC). **Bold** indicates the best performance.

2021). During inference, we set the decoding temperature to $T = 1.0$ and $\text{top-}p = 0.6$. We report **Avg@16 Accuracy**, computed as the mean accuracy over 16 sampled responses per problem, and **Average Tool Calls (ATC)**. To assess robustness under restricted tool access, we conduct a budget sensitivity analysis by varying the maximum allowed tool calls $B \in \{0, 1, 2, 3, 4\}$.

4.2 Main Results

AdaTIR Maintains Competitive Accuracy with Fewer Tool Invocations. Results in Table 1 and Figure 4 show that AdaTIR shifts the accuracy-efficiency trade-off, with accuracy margins ranging from -2.4% to $+3.3\%$ and ATC reductions from 22.1% to 97.6%. On AIME 2024 and 2025, AdaTIR-7B keeps accuracy close to or above GRPO while reducing tool use; on AIME 2024, accuracy increases from 33.8% to 37.1% and ATC decreases from 2.02 to 1.45. The reduction is larger on GSM8K, where AdaTIR reaches ATC 0.02 compared with 0.83 for GRPO, with a 1.8% accuracy gain. These results indicate that AdaTIR can reduce many tool calls while preserving competitive task accuracy.

No-tool Budget Robustness. Budget sensitivity analysis (Figure 4) evaluates whether the learned policy remains useful when tool access is restricted. The gap is most evident at **B0**, where tool calls are denied by the environment (see Appendix A.3). Under this no-tool setting, AdaTIR improves AIME 2024 accuracy by $+4.8\%$ absolute over the baseline. This shows that the learned policy remains more effective when external tools are unavailable, beyond merely reducing tool-call frequency when

tools are present. This result directly establishes improved robustness under a zero-tool budget. It does not, by itself, identify a mechanistic form of parameter-level reasoning internalization; avoiding the tool-call format and continuing in natural language is an alternative explanation.

AdaTIR also reduces cognitive offloading on simpler steps. While the baseline still invokes tools on GSM8K (ATC 0.83 at $B = 4$), AdaTIR reaches higher accuracy with near-zero ATC (0.02). A related trend appears outside the main Python-math setting: on Search-R1 QA tasks, AdaTIR improves average EM from 0.300 to 0.313 while reducing ATC from 1.00 to 0.95 (Appendix B); preliminary Llama and mixed-tool results show similar reductions in tool use (Appendix C).

5 Analysis

5.1 Comparison with OTC-style Reward Shaping

Experimental Setup for Direct Comparison with OTC-style Reward Shaping. To isolate whether difficulty awareness improves over a uniform tool-use penalty, we compare AdaTIR with an OTC-style reward implemented under the same Qwen2.5-7B-Instruct setting used in the main experiments. All methods use $N_{max} = 4$ and the same evaluation benchmarks, so the comparison directly measures the accuracy-efficiency trade-off under a matched tool budget.

AdaTIR Reduces the Accuracy Cost of Tool Penalties. As shown in Table 2, the OTC-style baseline reduces ATC more strongly than AdaTIR on several benchmarks, but it also lowers accuracy on every benchmark compared with GRPO.

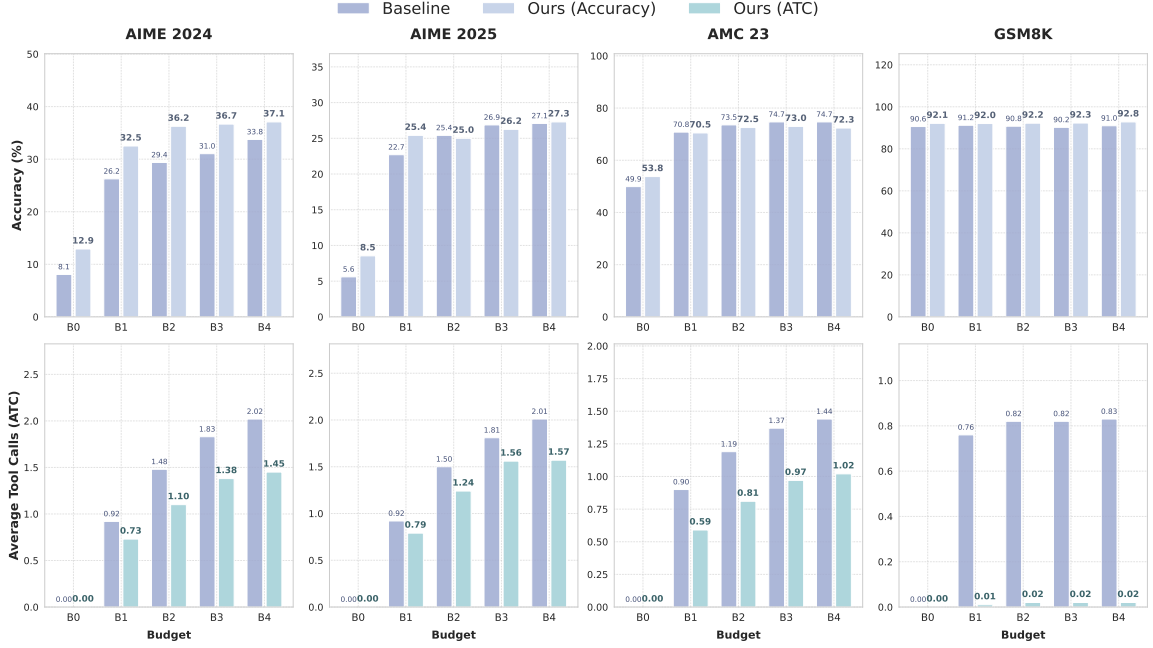


Figure 4: **Reasoning accuracy and tool invocation under varying tool budgets B .** The horizontal axis denotes the maximum allowed tool calls, ranging from B0 ($B = 0$) to B4 ($B = 4$). The top row presents reasoning accuracy (Avg@16), while the bottom row shows Average Tool Calls (ATC). AdaTIR usually uses fewer tool calls at comparable accuracy across budgets. At B0, where tool invocation is not permitted, AdaTIR obtains higher accuracy than the baseline, demonstrating improved no-tool budget robustness.

Method	N_{max}	AIME 24		AIME 25		AMC23		GSM8K	
		Acc ($\uparrow$)	ATC ($\downarrow$)	Acc ($\uparrow$)	ATC ($\downarrow$)	Acc ($\uparrow$)	ATC ($\downarrow$)	Acc ($\uparrow$)	ATC ($\downarrow$)
GRPO	4	33.8	2.02	27.1	2.01	74.7	1.44	91.0	0.83
OTC-style	4	29.6	1.09	25.3	1.16	69.2	0.85	89.1	0.05
AdaTIR (Ours)	4	37.1	1.45	27.3	1.57	72.3	1.02	92.8	0.02

Table 2: Matched comparison with an OTC-style reward under the main experimental configuration on Qwen2.5-7B-Instruct. All methods use the same tool-call limit ($N_{max} = 4$). OTC-style training obtains lower ATC on several benchmarks but loses accuracy relative to GRPO; AdaTIR gives a more balanced accuracy-efficiency trade-off in this setting.

Method	AIME 2024		AIME 2025	
	EM ($\uparrow$)	ATC ($\downarrow$)	EM ($\uparrow$)	ATC ($\downarrow$)
Vanilla GRPO	26.7	0.90	26.7	0.97
w/ Reward Shaping	30.0	0.33	20.0	0.27
w/ Clipped Advantage Shaping	33.3	0.50	26.7	0.70

Table 3: Ablation of CAS on AIME benchmarks comparing Exact Match (EM) and Average Tool Calls (ATC) against difficulty-aware reward shaping.

AdaTIR keeps accuracy close to or above GRPO on AIME 2024, AIME 2025, and GSM8K while still reducing tool calls. This supports the central design choice: tool-use pressure should be conditioned on estimated difficulty rather than applied uniformly.

5.2 Ablation Study of CAS: Ensuring Training Stability

To assess the role of **Clipped Advantage Shaping (CAS)**, we compare it with the **difficulty-aware reward shaping** approach formulated in Section 3.1. As illustrated in Figure 5, CAS makes efficiency learning more stable under group-normalized RL in this setting. The ablation uses the same constrained AIME setup across variants.

CAS Stabilizes Training in the Ablation. Without advantage clipping, trajectories with unusual tool-call counts can generate disproportionately large advantage signals. As shown in the left panel of Figure 5, the difficulty-aware reward shaping baseline shows a sharp gradient spike near step

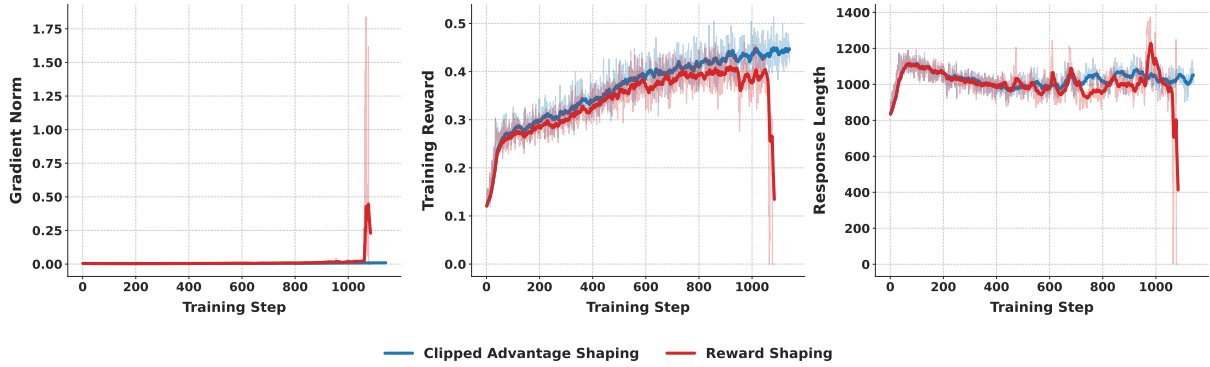


Figure 5: Training stability analysis of AdaTIR compared with standard reward shaping. From left to right, the panels display the **Gradient Norm**, **Training Reward**, and **Response Length** over training steps. Solid lines represent the smoothed curves using exponential moving average (EMA), while the semi-transparent shaded areas indicate the raw fluctuations.

Setting	Baseline		AdaTIR	
	Acc / EM	ATC	Acc / EM	ATC
Search-R1 QA avg.	0.300	1.00	0.313	0.95
Llama-3.2-3B avg.	20.08	1.23	21.12	1.02
Mixed-tool avg.	20.11	1.77	19.99	1.01

Table 4: Compact generalization summary beyond the main Qwen2.5 Python-math setting. Search-R1 reports average EM over seven QA datasets; Llama-3.2-3B and mixed-tool rows report average accuracy over AIME24, AIME25, and AMC23.

1,100 and then collapses. In contrast, the CAS curve remains stable in this run. This pattern is also reflected in Table 3: direct reward shaping reduces ATC more aggressively, but the reduction coincides with an accuracy drop on AIME 2025. CAS provides a more moderate efficiency signal and avoids this collapse in the evaluated setup.

CAS Balances Efficiency and Accuracy in the Ablation. The results in Table 3 show that CAS also keeps stronger reasoning accuracy in this ablation. Compared with direct reward shaping, CAS achieves a higher Exact Match (EM) score on AIME 2024 (33.3% vs. 26.7%) and avoids the AIME 2025 drop (26.7% vs. 20.0%). A plausible explanation is that the clipped auxiliary signal reduces over-suppression of tool calls, leaving the model room to use external computation on harder steps.

5.3 Generalization Beyond the Main Python-Math Setting

Preliminary Evidence Beyond the Main Python-Math Setting. The main math benchmarks provide the primary evidence for the accuracy-

efficiency trade-off. Table 4 summarizes three additional settings that test whether the same mechanism remains useful outside the main configuration. On Search-R1 QA tasks, AdaTIR improves average EM from 0.300 to 0.313 while reducing ATC from 1.00 to 0.95, suggesting that difficulty-aware suppression can help even when external knowledge is often needed. On a second model family, Llama-3.2-3B, AdaTIR improves average accuracy from 20.08 to 21.12 and reduces ATC from 1.23 to 1.02. In a heterogeneous tool environment with Python, Web Search, Wolfram Alpha, and OEIS, AdaTIR cuts ATC from 1.77 to 1.01 while keeping average accuracy nearly unchanged (20.11 $\rightarrow$ 19.99), with gains on AIME25 and a small drop on AMC23.

What These Generalization Results Do and Do Not Show. We do not treat these experiments as full-scale scaling studies. The Llama and mixed-tool runs use only 100 training steps, and Search-R1 evaluation follows the original early-checkpoint protocol because that training recipe tends to collapse after roughly 200 steps. The right reading is therefore directional rather than definitive: across a different domain, a different backbone, and a different toolset, AdaTIR reduces tool reliance without clear evidence that the mechanism is tied to a single model family or a Python-only interface.

5.4 Case Study

Figure 3 compares a representative reasoning process between the TIR baseline and AdaTIR. The baseline relies on external execution; after the symbolic solver fails, it attempts an inappropriate brute-force search and returns an incorrect answer. In contrast, AdaTIR derives the relevant relationship

Category	GSM8K		AIME	
	Base.	Ours	Base.	Ours
Missing Tool-use	32.8	91.6	8.1	26.5
Tool-induced Noise	9.2	1.1	57.7	38.2
Reasoning Failure	58.0	7.4	34.2	35.3

Table 5: Failure-mode distribution comparing the TIR baseline and AdaTIR. Percentages are computed over failed examples only.

through algebraic reasoning without external tool calls. Additional case studies and categorical error analyses are provided in Appendix D and Appendix E.

Failure Reduction Comes with a Clearer Failure Mode. Table 5 makes the trade-off concrete. On GSM8K, *Tool-induced Noise* decreases from 9.2% to 1.1%, while most residual errors shift to *Missing Tool-use*. On AIME, *Tool-induced Noise* drops from 57.7% to 38.2%, while *Missing Tool-use* rises from 8.1% to 26.5%. This shift is important for interpreting the method. AdaTIR is not simply making models avoid tools; it reduces tool-induced distraction but can increase failures where internal reasoning is insufficient.

The error analysis therefore reveals a limitation of the difficulty proxy. AdaTIR reduces tool-induced noise, but it can increase *Missing Tool-use* errors when a task appears easy under tool-available rollouts but is harder to solve internally. This follows from the estimator: ϕ_q measures difficulty under the current TIR policy, not intrinsic no-tool difficulty. We discuss representative failures in Appendix E.

6 Related Works

RL for Tool-Integrated Reasoning Recent research uses Reinforcement Learning (RL) to enable LLMs to invoke external tools. In mathematical reasoning, ToRL (Li et al., 2025c), ReTool (Feng et al., 2025), and Effective CIR (Bai et al., 2025) train models to use Python interpreters for complex tasks, while zero-resource methods like SimpleTIR (Xue et al., 2025) and ZeroTIR (Mai et al., 2025) explore end-to-end reasoning without extensive supervised fine-tuning. In more general settings, agents such as Search-R1 (Jin et al., 2025) and WebSailors (Li et al., 2025a) use search engines to access external knowledge. These systems improve tool-augmented reasoning, but correctness-focused training can still

induce redundant tool invocations (Wang et al., 2025a), increasing inference overhead and exposing the model to irrelevant feedback.

Efficiency for Tool-Integrated Reasoning Research explicitly targeting efficiency in tool-integrated reasoning remains limited. Some prompt-guided frameworks calculate confidence scores (Wang et al., 2025b; Li et al., 2025b; Shen et al., 2024; Qian et al., 2025) or prompt models to account for tool costs (Liu et al., 2025), but they often rely on expert templates. Recent work has shifted toward training-based methods (Wang et al., 2025a; Hashemi et al., 2025). OTC (Wang et al., 2025a), for example, uses reward shaping to improve tool productivity. However, a uniform penalty across task difficulties can reduce necessary tool use on harder questions. AdaTIR addresses this issue by conditioning efficiency pressure on an online estimate of task difficulty.

7 Conclusion

We propose AdaTIR, a difficulty-aware policy optimization framework for adaptive tool use. By combining a difficulty-aware efficiency reward with Clipped Advantage Shaping (CAS), AdaTIR reduces cognitive offloading while keeping correctness as the primary learning signal. Experiments on mathematical benchmarks show that AdaTIR reduces tool calls by up to 97.6% while preserving competitive accuracy. Additional QA and mixed-tool results provide preliminary evidence that the approach can extend beyond the main Python-math setting.

Limitations

While AdaTIR improves no-tool budget robustness, it has practical limitations. First, the main evaluation focuses on mathematical reasoning. We include Search-R1 QA results and preliminary cross-family and mixed-tool experiments in the appendix, but broader validation on open-ended coding and long-horizon agent tasks remains future work. Second, the current difficulty estimator is a policy-dependent proxy: it measures success under tool-available rollouts, not intrinsic no-tool difficulty. Under distribution shift, unusual or underrepresented examples may therefore be misclassified as easy. This can cause Missing Tool-use failures when a problem is easy with tools but brittle under internal reasoning. Third, we did not exhaustively tune ϕ_{low} , γ , or δ across model scales and

domains. More adaptive difficulty estimation and hyperparameter selection may further improve the accuracy-efficiency frontier. Fourth, ATC counts invocations but does not measure heterogeneous tool latency, generated tokens, wall-clock time, or monetary cost; practical efficiency claims should therefore be read as tool-call efficiency rather than end-to-end system efficiency. Finally, suppressing tools can be harmful when external computation or retrieval provides necessary verification. This risk is especially important in high-stakes or out-of-distribution settings, where confident internal reasoning may be unsupported.

References

- Fei Bai, Yingqian Min, Beichen Zhang, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. 2025. [Towards effective code-integrated reasoning](#). *Preprint*, arXiv:2505.24480.
- Bytedance-Seed-Foundation-Code-Team, :, Yao Cheng, Jianfeng Chen, Jie Chen, Li Chen, Liyu Chen, Wentao Chen, Zhengyu Chen, Shijie Geng, Aoyan Li, Bo Li, Bowen Li, Linyi Li, Boyi Liu, Jiaheng Liu, Kaibo Liu, Qi Liu, Shukai Liu, and 37 others. 2025. [Fullstack bench: Evaluating llms as full stack coders](#). *Preprint*, arXiv:2412.00535.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *Preprint*, arXiv:2110.14168.
- Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. 2025. [The entropy mechanism of reinforcement learning for reasoning language models](#). *Preprint*, arXiv:2505.22617.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. [Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning](#). *Preprint*, arXiv:2501.12948.
- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. [Retool: Reinforcement learning for strategic tool use in llms](#). *Preprint*, arXiv:2504.11536.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujia Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. [Tora: A tool-integrated reasoning agent for mathematical problem solving](#). In *International Conference on Representation Learning*, volume 2024, pages 48362–48395.
- Helia Hashemi, Victor Rühle, and Saravan Rajmohan. 2025. [Cost-aware retrieval-augmentation reasoning models with adaptive retrieval depth](#). *Preprint*, arXiv:2510.15719.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. [Swe-bench: Can language models resolve real-world github issues?](#) *Preprint*, arXiv:2310.06770.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. [Search-rl: Training llms to reason and leverage search engines with reinforcement learning](#). *arXiv preprint arXiv:2503.09516*.
- Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *Advances in neural information processing systems*, 12.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. [Natural questions: A benchmark for question answering research](#). *Transactions of the Association for Computational Linguistics*, 7:452–466.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. [Websailor: Navigating super-human reasoning for web agent](#). *Preprint*, arXiv:2507.02592.
- Wenjun Li, Dexun Li, Kuicai Dong, Cong Zhang, Hao Zhang, Weiwen Liu, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025b. [Adaptive tool use in large language models with meta-cognition trigger](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13346–13370, Vienna, Austria. Association for Computational Linguistics.
- Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025c. [Torl: Scaling tool-integrated rl](#). *Preprint*, arXiv:2503.23383.
- Heng Lin and Zhongwen Xu. 2025. [Understanding tool-integrated reasoning](#). *Preprint*, arXiv:2508.19201.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. [Lost in the middle: How language models use long contexts](#). *Transactions of the Association for Computational Linguistics*, 12:157–173.

- Tengxiao Liu, Zifeng Wang, Jin Miao, I-Hung Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, Samira Daruki, Yi Liang, William Yang Wang, Tomas Pfister, and Chen-Yu Lee. 2025. [Budget-aware tool-use enables effective agent scaling](#). *Preprint*, arXiv:2511.17006.
- Xinji Mai, Haotian Xu, Xing W, Weinong Wang, Yingying Zhang, and Wenqiang Zhang. 2025. [Agentic RL scaling law: Spontaneous code execution for mathematical problem solving](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. [Openai o1 system card](#). *Preprint*, arXiv:2412.16720.
- Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiushi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. [SMART: Self-aware agent for tool overuse mitigation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4604–4621, Vienna, Austria. Association for Computational Linguistics.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, and 25 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *Preprint*, arXiv:2402.03300.
- Yuanhao Shen, Xiaodan Zhu, and Lei Chen. 2024. [SMARTCAL: An approach to self-aware tool-use evaluation and calibration](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 774–789, Miami, Florida, US. Association for Computational Linguistics.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. [Hybridflow: A flexible and efficient rlhf framework](#). In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 1279–1297. ACM.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: Language agents with verbal reinforcement learning](#). *Preprint*, arXiv:2303.11366.
- Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. [Agentic reasoning and tool integration for llms via reinforcement learning](#). *Preprint*, arXiv:2505.01441.
- Hongru Wang, Cheng Qian, Wanjun Zhong, Xiushi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025a. [Acting less is reasoning more! teaching model to act efficiently](#). *Preprint*, arXiv:2504.14870.
- Hongru Wang, Boyang Xue, Baohang Zhou, Tianhua Zhang, Cunxiang Wang, Huimin Wang, Guanhua Chen, and Kam-Fai Wong. 2025b. [Self-DC: When to reason and when to act? self divide-and-conquer for compositional unknown questions](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6510–6525, Albuquerque, New Mexico. Association for Computational Linguistics.
- Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023. [Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning](#). *Preprint*, arXiv:2310.03731.
- Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. 2025c. [Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning](#). *Preprint*, arXiv:2506.01939.
- Nan Xu, Fei Wang, Ben Zhou, Bangzheng Li, Chaowei Xiao, and Muhao Chen. 2024. [Cognitive overload: Jailbreaking large language models with overloaded logical thinking](#). In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3526–3548, Mexico City, Mexico. Association for Computational Linguistics.
- Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. [Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning](#). *Preprint*, arXiv:2509.02479.

A Implementation Details

A.1 Implementation Details for Main Experiments

The experimental settings follow ReTool (Feng et al., 2025). The main experiments are conducted using Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct backbones. Following the data configuration in ReTool, the training data for the cold-start SFT phase and RL phase are sourced from specialized mathematical reasoning datasets.

For the reinforcement learning phase, the difficulty-aware reward and CAS mechanism use the following key parameters: the difficulty threshold ϕ_{low} is set to 0.8 to trigger efficiency pressure for low-difficulty tasks; the CAS scale γ is set to 0.9 to regulate efficiency incentives; the advantage clipping ratio δ is set to 0.99; and the clipping offset η is set to 10^{-4} . The remaining training and evaluation hyperparameters are summarized in Table 6.

Hyperparameter	Value
Group Size (G)	16
Max Sequence Length	4096
Training batch size	64
PPO mini batch size	16
PPO clip ratio (low / high)	0.2 / 0.28
KL coefficient	0.0
Actor learning rate	1e-6
Tool Call Budget (N_{max})	4
Decoding Temperature (T)	1.0
Top- p Sampling	0.6

Table 6: Summary of experimental hyperparameters for main experiments.

A.2 Implementation Details for Comparison with OTC-style Reward Shaping

For the matched OTC-style comparison in Table 2, we use the same Qwen2.5-7B-Instruct backbone, training configuration, benchmarks, and tool-call limit ($N_{max} = 4$) as the main experiments. This isolates the effect of uniform tool-use penalization from AdaTIR’s difficulty-aware penalization. All other reward and CAS parameters remain consistent with the main experiments.

A.3 Prompts and Templates

For comparability and reproducibility, we follow the prompt templates and system instructions provided by ReTool (Feng et al., 2025). The full instruction template used during both SFT and RL phases is presented in Table 13.

Feedback when Exceeding Budget: When the number of tool calls exceeds the specified budget N_{max} during reasoning, any further tool call attempts will receive the following feedback: **"Tool call budget exceeded; the code was not executed. Please continue reasoning and give the final answer."** This encourages the model to transition from external tool invocation to internal reasoning during the RL phase.

B Generalization Results on Search R1

To test whether AdaTIR transfers to Question Answering (QA), we apply it to search-based workflows. We choose **Search-R1** (Jin et al., 2025) as the baseline and follow its original experimental settings. Specifically, we apply GRPO on the Qwen2.5-3B model, and all hyperparameters introduced by AdaTIR are kept identical to those described in our main implementation (Section A). This comparison tests whether the same efficiency mechanism remains useful in a search-heavy environment.

As summarized in Table 7, AdaTIR improves average EM by 1.3 points while reducing ATC from 1.00 to 0.95. This result is consistent with the difficulty-aware policy reducing redundant search on simpler questions, although the gain is modest compared with the math setting.

We hypothesize that the smaller efficiency gain compared with mathematical tasks stems from two factors. First, following the observation in the Search-R1 study, models tend to collapse after approximately 200 training steps (Jin et al., 2025). Consequently, both Search-R1 and AdaTIR use the 200-step checkpoint for evaluation. At this early stage of training, the correctness advantage may remain more dominant than the efficiency penalty. Second, QA tasks often require external knowledge, which limits the potential for further budget compression in search-based workflows.

C Additional Generalization Results

We additionally evaluate whether AdaTIR transfers beyond the main Qwen2.5 Python-math setting. These experiments use shorter training runs and should be interpreted as preliminary evidence rather than exhaustive scaling results.

D Extended Case Studies

As illustrated in Figure 6, we observe **cognitive offloading**, where the baseline delegates trivial arithmetic to external tools despite having resolved the logic internally. This behavior is consistent with a policy that does not distinguish easy from hard intermediate steps. AdaTIR reduces this pattern by encouraging internal reasoning on tasks that the current policy already solves reliably.

Furthermore, Figure 7 shows how incorrect or unexpected tool feedback can distract the model. When a tool returns a wrong or unhelpful result, the baseline may shift from solving the original

Table 7: Comparison on Search-R1 generalization tasks using **Qwen2.5-3b-Base** as the backbone. Each cell reports **EM / ATC**, where EM is the Exact Match accuracy and ATC is the Average Tool Calls. **Bold** indicates the best EM, and **green** indicates reduced tool calls.

Method	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	Avg.
Search-R1	0.391 / 1.00	0.582 / 1.00	0.392 / 1.00	0.285 / 1.00	0.267 / 1.01	0.054 / 1.00	0.121 / 1.00	0.300 / 1.00
AdaTIR	0.403 / 0.94	0.597 / 0.92	0.416 / 0.93	0.297 / 0.94	0.260 / 0.98	0.076 / 0.99	0.145 / 0.96	0.313 / 0.95

Model	Method	Steps	AIME24		AIME25		AMC23	
			Acc	ATC	Acc	ATC	Acc	ATC
Llama-3.2-3B	GRPO	100	8.13	1.26	14.16	1.25	37.96	1.19
Llama-3.2-3B	AdaTIR	100	9.58	1.02	15.83	1.02	37.96	1.01

Table 8: Preliminary cross-family results on Llama-3.2-3B. AdaTIR reduces tool calls while improving or matching accuracy across the evaluated benchmarks.

Toolset	Steps	Method	AIME24		AIME25		AMC23	
			Acc	ATC	Acc	ATC	Acc	ATC
Python+Search+Wolfram+OEIS	100	GRPO	11.25	1.80	16.74	1.73	32.34	1.78
Python+Search+Wolfram+OEIS	100	AdaTIR	11.25	1.00	17.63	1.03	31.09	1.00

Table 9: Preliminary mixed-tool results with Python, Web Search, Wolfram Alpha, and OEIS. AdaTIR reduces tool use and matches or improves accuracy on two of three benchmarks, with a small accuracy drop on AMC23.

problem to debugging the tool call. This creates a logical interruption: the model spends tokens managing tool output rather than completing the reasoning chain. AdaTIR reduces this failure mode in the observed cases by relying more on internal reasoning before invoking tools.

E Error Analysis

To understand the strategic differences between the baseline and our model, we categorize failures into three distinct modes. This analysis focuses on AIME (hard) and GSM8K (easy) to contrast performance across different complexity levels. We exclude cases where both models fail due to fundamental capacity limits.

- **Missing Tool-use:** The model relies on internal reasoning but fails because the calculation or logic exceeds its current parametric precision. This is often a sign of over-confidence in its internal reasoning. Table 12 shows an incorrect no-tool trajectory together with its verified correction, making the unverified state-tracking error explicit.
- **Tool-induced Noise:** Incorrect or unexpected tool feedback distracts the model, causing a logical interruption where it shifts focus to

debugging code instead of the main problem (see Figure 7 for a detailed case).

- **Reasoning Failure:** Pure logical errors that occur independently of whether a tool was used or how it responded. Table 11 shows how an incorrect endpoint formalization produces a wrong answer even though the tool executes the submitted program correctly, followed by a verified correction.

Tables 11 and 12 are hand-selected illustrations of the two failure modes, not representative samples or evidence for the category frequencies reported below. We show each failed model output together with a verified answer and a diagnosis of the error.

Category	GSM8K (Easy)		AIME (Hard)	
	Base.	Ours	Base.	Ours
Missing Tool-use	32.8	91.6	8.1	26.5
Tool-induced Noise	9.2	1.1	57.7	38.2
Reasoning Failure	58.0	7.4	34.2	35.3

Table 10: Comparison of failure mode distributions (%) between the baseline and our model.

As shown in Table 10, there is a clear shift in how the models fail. On **GSM8K**, most AdaTIR errors (91.6%) fall under *Missing Tool-use*. This suggests that the model often relies on internal reasoning, but sometimes overestimates its ability to handle specific calculations without tools. At the same time, *Tool-induced Noise* drops to 1.1%.

On the more difficult **AIME** tasks, the baseline has a high share of *Tool-induced Noise* errors (57.7%). This supports the observation that frequent tool interactions can distract the model when tool outputs are wrong or unhelpful. AdaTIR reduces this category to 38.2%, but its *Missing Tool-use* rate increases to 26.5%. This is a real trade-off: the model is less often derailed by external feedback, but more often fails when internal reasoning is not enough.

F Additional Analysis of the Method

F.1 Choice of the Sine-based Reward Function

We choose a sine-based reward instead of a simple linear one because its non-linear shape gives stronger pressure against early tool calls. Specifically, the sine curve is steeper when the number of tool calls K is small, which places a higher penalty on the first few calls. This encourages the model to solve simple tasks through internal reasoning rather than calling tools by default. The smoothness of the sine function also avoids the discontinuities of step-based rewards.

F.2 Understanding the CAS Mechanism

Clipped Advantage Shaping (CAS) is designed to keep the efficiency term secondary to the correctness term. The formulation in Eq. 10 is:

$$A_i^{\text{CAS}} = A_i^{\text{acc}} + r_{\text{acc},i} \cdot \gamma \cdot \text{clip}(A_i^{\text{eff}}, -\delta|A_i^{\text{acc}}| - \eta, \delta|A_i^{\text{acc}}| + \eta) \quad (12)$$

where $r_{\text{acc},i} \in \{0, 1\}$ is the correctness mask, $\gamma > 0$ is the efficiency weight, and $\delta \in [0, 1), \eta > 0$ are clipping parameters.

Scenario 1: Incorrect Trajectories ($r_{\text{acc},i} = 0$).

For incorrect responses, the mask zeros out the efficiency term: $A_i^{\text{CAS}} = A_i^{\text{acc}}$. Since A_i^{acc} is typically negative for incorrect answers, CAS leaves the main negative feedback unchanged.

Scenario 2: Correct Trajectories ($r_{\text{acc},i} = 1$).

Here, the effect of the clipped efficiency term depends on the group distribution:

- **Case A: Mixed Group** ($A_i^{\text{acc}} > 0$). When the group contains both correct and incorrect responses, correct trajectories typically yield positive accuracy advantages. The worst-case advantage occurs when the efficiency penalty is maximal:

$$A_i^{\text{CAS}} \geq A_i^{\text{acc}} + \gamma \cdot (-\delta|A_i^{\text{acc}}| - \eta) \quad (13)$$

$$= (1 - \gamma\delta)A_i^{\text{acc}} - \gamma\eta \quad (14)$$

Sufficient condition. A positive CAS advantage is obtained when $A_i^{\text{acc}} > \frac{\gamma\eta}{1-\gamma\delta}$. With our default settings ($\gamma = 0.9, \delta = 0.99, \eta = 1\text{e-}4$), the denominator $1 - \gamma\delta \approx 0.109$, giving a threshold of $A_i^{\text{acc}} \approx 8.2\text{e-}4$. In our runs, normalized advantages for correct answers in mixed groups are usually much larger than

this threshold (e.g., > 0.1), so sign reversal is not observed in this regime. This is a condition-level argument rather than a guarantee for all possible group statistics.

- **Case B: All-Correct Group** ($A_i^{\text{acc}} = 0$). If all trajectories in a group are correct, $A_i^{\text{acc}} = 0$. In this convergence state, the clipping range shrinks to $[-\eta, \eta]$:

$$A_i^{\text{CAS}} = 0 + \gamma \cdot \text{clip}(A_i^{\text{eff}}, -\eta, \eta) \geq -\gamma\eta \quad (15)$$

Here, the negative contribution is bounded by $-\gamma\eta$ (approx. $-9\text{e-}5$). This small magnitude lets efficiency rank correct answers while limiting the size of negative updates.

Case Study: Cognitive Offloading in Simple Tasks

Question: A candle melts by 2 centimeters every hour that it burns. How many centimeters shorter will a candle be after burning from 1:00 PM to 5:00 PM?

Baseline: Inefficiency via Cognitive Offloading

Thought Trace: Alright, let's tackle this problem step by step. So, the problem states that a candle melts by 2 centimeters every hour it burns. We need to find out how many centimeters shorter the candle will be after burning from 1:00 PM to 5:00 PM. [...] So, if per hour it melts 2 cm, then in 4 hours, it would melt $2 \text{ cm/hour} * 4 \text{ hours} = 8 \text{ cm}$. Wait, let me verify that. Let's use code to calculate the total melting.

```
# Redundant tool call for simple multiplication: 4 * 2
hours = 5 - 1 # From 1:00 PM to 5:00 PM inclusive
melting_per_hour = 2 # cm
total_melting = hours * melting_per_hour
print(total_melting)
```

Response: 8

The code execution result confirms that the total melting is 8 cm. So, the candle will be 8 centimeters shorter.

Tool Calls: 1

Final Answer: 8 ✓

Ours: Efficiency via Internal Reasoning

Thought Trace: Okay, let me try to solve this problem. Hmm, so a candle melts by 2 centimeters every hour that it burns. [...] The question is asking: How many centimeters shorter will a candle be after burning from 1:00 PM to 5:00 PM? Hmm, so that's a 4-hour period, right? From 1:00 PM to 5:00 PM is 4 hours. So if the candle melts at a rate of 2 centimeters per hour, then over 4 hours, it would melt $2 \text{ cm/hour} * 4 \text{ hours} = 8 \text{ centimeters}$. So the candle would be 8 centimeters shorter after 4 hours. Hmm, that seems straightforward. I don't think I missed anything here.

Tool Calls: 0

Final Answer: 8 ✓

Figure 6: Comparison of reasoning efficiency under redundant tool use. In this example, our model relies on internal logic and avoids the unnecessary cognitive offloading observed in the baseline.

Case Study: Reasoning Impairment under Tool-Induced Noise

Question: Find the number of ways to place a digit in each cell of a 2x3 grid so that the sum of the two numbers formed by reading left to right is 999, and the sum of the three numbers formed by reading top to bottom is 99. The grid below is an example of such an arrangement because $8 + 991 = 999$ and $9 + 9 + 81 = 99$.

Baseline: Logic Erosion via Repeated Tool-Execution Noise

Thought Trace: Alright, let's tackle this problem step by step. [...] Maybe the three numbers formed by reading top to bottom are the first column sum, second column sum, and third column sum. Then, their sum is 99. [...] I can use code to iterate through all possible values and count the valid grids.

```
# First Attempt: Brute-forcing with column sum logic (a+d+b+e+c+f == 99)
count = 0
for a in range(1, 10):
    for b in range(0, 10):
        for c in range(0, 10):
            for d in range(1, 10):
                for e in range(0, 10):
                    for f in range(0, 10):
                        if (100*a+10*b+c) + (100*d+10*e+f) == 999:
                            if (a+d) + (b+e) + (c+f) == 99: count += 1
print(count)
```

Response: 0

Hmm, the code execution result is 0. That suggests there might be an issue with the logic. [...] Let me adjust the code to print out the valid grids so I can verify.

```
valid_grids = []
for a, b, c, d, e, f in itertools.product(range(1,10), range(10), ...):
    if sum_lr == 999 and col_sum == 99: valid_grids.append((a,b,c,d,e,f))
print(len(valid_grids))
```

Response: 0

The code execution result is still 0. [...] Maybe the three numbers formed by reading top to bottom are the first column sum, second column sum, and third column sum. [...] Wait, maybe I should write code to generate all possible grids (digits 0-9 for each cell):

```
# Third Attempt: Repeating the same flawed logic with minor cell range adjustments
count = 0
for a in range(1, 10):
    # ... (similar nested loops)
    if num1 + num2 == 999 and col_sum == 99: count += 1
print(count)
```

Response: 0

The code execution result is 0. This means there are no valid grids that satisfy both conditions. Therefore, the answer is 0.

Tool Calls: 3

Final Answer: 0 ✖

Ours: Precise Formalization for Reliable Reasoning

Thought Trace: [...] Reading top to bottom, the three numbers would be $(10a + d)$, $(10b + e)$, and $(10c + f)$. Their sum should be 99. [...] This seems manageable. Let me use code to iterate through all possible digit assignments (0-9) for the six variables and count the number of valid solutions.

```
import itertools
valid_count = 0
for a, b, c, d, e, f in itertools.product(range(10), repeat=6):
    sum_lr = (100 * a + 10 * b + c) + (100 * d + 10 * e + f)
    sum_tb = (10 * a + d) + (10 * b + e) + (10 * c + f)
    if sum_lr == 999 and sum_tb == 99: valid_count += 1
print(valid_count)
```

Response: 45

The code returns 45 as the number of valid grids. Therefore, the number of ways is 45.

Tool Calls: 1

Final Answer: 45 ✔

Figure 7: Comparison of reasoning under tool-induced noise. Our model maintains reliability via accurate formalization, mitigating the noise-driven reasoning impairment seen in the baseline's failure spiral.

Illustrative Reasoning Failure with a Verified Correction

Question: Consider the paths of length 16 that follow the lines from the lower left corner to the upper right corner on an 8×8 grid. Find the number of such paths that change direction exactly four times.

Incorrect model trajectory (abridged). The model treats the upper-right corner as $(7, 7)$, notices that this conflicts with a length-16 path, and then enumerates all strings in $\{R, U\}^{16}$ having four direction changes. The submitted program returns 2730, which the model accepts as the final answer.

Diagnosis. The external tool executes the submitted enumeration correctly, but the model’s formalization omits the endpoint constraint. An 8×8 grid has endpoints $(0, 0)$ and $(8, 8)$, so every valid monotone path must contain exactly eight R steps and eight U steps. The failure is therefore attributed to reasoning rather than tool-induced noise.

Verified correction. Four direction changes divide a path into five nonempty runs. A path starting with R has three R -runs and two U -runs, whose positive lengths can be chosen in $\binom{7}{2}\binom{7}{1} = 147$ ways. Starting with U gives the symmetric count. Hence the verified answer is

$$2 \binom{7}{2} \binom{7}{1} = 294.$$

Model Tool Calls: 2

Model Answer: 2730 ✗

Verified Answer: 294 ✓

Table 11: A hand-selected **Reasoning Failure**. The tool correctly executes an incorrectly formalized enumeration; the verified correction enforces the endpoint constraint. This example is illustrative rather than representative.

Illustrative Missing Tool-use Failure with a Verified Correction

Question: Alice and Bob play the following game. A stack of n tokens lies before them. The players take turns with Alice going first. On each turn, the player removes either 1 token or 4 tokens from the stack. Whoever removes the last token wins. Find the number of positive integers n less than or equal to 2024 for which there exists a strategy for Bob that guarantees that Bob will win the game regardless of Alice’s play.

Incorrect model trajectory (abridged). Without invoking a tool, the model sets $f(0) = L$, $f(1) = W$, and $f(2) = L$, but then incorrectly labels $f(3) = L$ even though removing one token reaches the losing state $f(2)$. It propagates this state-tracking error, claims that only multiples of five are losing, and finally reports the number of nonmultiples, $2024 - 404 = 1620$.

Diagnosis. The model neither checks the recurrence nor verifies the resulting pattern with an available tool. The initial logical error and the inconsistent final counting step therefore remain undetected, illustrating Missing Tool-use in this taxonomy.

Verified correction. Let L denote a losing position. The initial states are $0 = L$, $1 = W$, $2 = L$, $3 = W$, and $4 = W$. The pattern is periodic: the losing positions are exactly $n \equiv 0$ or $2 \pmod{5}$. For every positive $n \equiv 0 \pmod{5}$, either legal move reaches residue 4 or 1; from residue 2, either move reaches residue 1 or 3. Conversely, residues 1, 3, and 4 can reach a losing residue in one move. Up to 2024, there are 404 positive multiples of five and 405 numbers congruent to 2, giving

$$404 + 405 = 809.$$

Model Tool Calls: 0

Model Answer: 1620 ✗

Verified Answer: 809 ✓

Table 12: A hand-selected **Missing Tool-use** failure. The model does not verify a faulty recurrence, while the verified correction identifies both losing residue classes. This example is illustrative rather than representative.

Template Prompt for RL Training
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 (wrapped in <code><interpreter>output_str</interpreter></code>) can be returned to aid your reasoning and help you arrive at the final answer. The Python code should be complete scripts, including necessary imports. Each code snippet is wrapped with <code><code>\n```python\ncode snippet\n```</code></code>. The last part of your response should be in the following format: <pre> <answer> \boxed{'The final answer goes here.'} </answer> *user question:* {question} Remember to place the final answer in the last part using the format: <answer> \boxed{'The final answer goes here.'} </answer> </pre>

Table 13: The full prompt template following the ReTool protocol (Feng et al., 2025). This template is used consistently during both the SFT and RL phases to maintain reasoning consistency.