

When to Trust Tools? Adaptive Tool Trust Calibration For Tool-Integrated Math Reasoning

Anonymous ACL submission

Abstract

Large reasoning models (LRMs) have achieved strong performance enhancement through scaling test time computation, but due to the inherent limitations of the underlying language models, they still have shortcomings in tasks that require precise computation and extensive knowledge reserves. Tool-Integrated Reasoning (TIR) has emerged as a promising paradigm that incorporates tool call and execution within the reasoning trajectory. Although recent works have released some powerful open-source TIR models, our analysis reveals that these models still suffer from critical deficiencies. We find that when the reasoning of the model conflicts with the tool results, the model tends to believe in its own reasoning. And there are cases where the tool results are correct but are ignored by the model, resulting in incorrect answers, which we define as “Tool Ignored”. This indicates that the model does not know when to trust or ignore the tool. To overcome these limitations, We introduce Adaptive Tool Trust Calibration (ATTC), a novel framework that guides the model to adaptively choose to trust or ignore the tool results based on the confidence score of generated code blocks. The experimental results from various open-source TIR models of different sizes and across multiple datasets demonstrate that ATTC effectively reduces the “Tool Ignored” issue, resulting in a performance increase of 4.1% to 7.5%.

1 Introduction

The rapid evolution of Large Reasoning Models (LRMs) (OpenAI et al., 2024; DeepSeek-AI et al., 2025a; Yang et al., 2025; Team et al., 2025; Comanici et al., 2025) represents a transformative milestone in the history of Large Language Models (LLMs). By scaling test-time computation (Ji et al., 2025), these models have achieved substantial performance gains in handling challenging reasoning tasks. Unlike conventional LLMs that typically generate responses in a direct, LRMs engage

Question

The smallest distance between the origin and a point on the graph of $y = \frac{1}{2}x^2 - 9$ can be expressed as a . Find a^2 .

Vanilla TIR's response

Reasoning
To find the smallest distance between the origin $((0,0))$. \langle omitted \rangle
The minimum value of (D^2) is 18. Therefore, $(a^2 = 18)$. Let's verify this with Python code to ensure the accuracy of the result.

Code Block
python
import sympy as sp
Define the variable
x = sp.symbols('x')
 \langle omitted \rangle
print(min_value)

Tool Call
output
17.000000000000000

Reasoning and Arrive at Final Answer
Let's carefully re-evaluate the steps again manually: \langle omitted \rangle The minimum value of (D^2) is 18, so $(a^2 = 18)$.
It seems the manual calculation was correct after all. The value of (a^2) is indeed (18) . Therefore, the final answer is:
 $\boxed{18}$

Figure 1: Case of the “Tool Ignored” phenomenon. The model arrive at an incorrect answer of 18 through reasoning, while the tool provide the correct result of 17. The model ignore the tool and give an incorrect answer.

in long Chain-of-Thought (CoT) reasoning before generating an answer. This shift toward a systematic deliberation process allows the model to refine its logic and arrive at more robust final outputs.

Nevertheless, LRMs are still constrained by the inherent limitations of the underlying language models (Zhao et al., 2025; Yue et al., 2025), most notably in areas requiring precise numerical computation and comprehensive knowledge coverage. To mitigate these limitations, Tool-Integrated Reasoning (TIR) (Gou et al., 2024; Wang et al., 2023; Liao et al., 2024) has emerged as a promising paradigm that incorporates tool call and execution within the reasoning trajectory. By incorporating external tools such as code executors and search engines, TIR empowers models to transcend the performance bottlenecks of purely reasoning.

Early works in TIR primarily rely on prompt

engineering (Wang et al., 2025; Yuan et al., 2024; Qian et al., 2024; Chen et al., 2023; Yang et al., 2024b) to guide LLMs in tool call. However, these approaches are heavily dependent on meticulously crafted prompts, which limit their scalability and generalizability. Some later works use supervised finetuning (Chen et al., 2025; Qian et al., 2025; Yang et al., 2024a; Yao et al., 2023; Wang et al., 2023) to internalize the behavior pattern of the model actively calling tools in reasoning by training models on specialized datasets enriched with tool call demonstrations. Nevertheless, SFT-based methodologies exhibit inherent limitations, as they constrain models to strictly adhere to the tool usage patterns present in the training data distribution, these models often fail to develop adaptive strategies. To address these issues, several recent works focus on applying reinforcement learning (Feng et al., 2025; Li et al., 2025; Jiang et al., 2025; Bai et al., 2025; Xue et al., 2025) to improve the tool use ability of models. These works enable the model to have more flexible strategies when calling tools based on the complexity of the task.

In this study, we specifically focus on the application of TIR in scenarios where code executors are utilized as the tool. Although existing works have enabled the models to perform TIR, our analysis reveals that existing open-source TIR models still suffer from critical deficiencies. Most notably, there remains a persistent struggle to achieve an optimal balance between external tool result and reasoning. By analyzing the reasoning trajectories of TIR models, we observe a widespread contradiction between the models’ reasoning and the results provided by external tools in false cases. When such conflicts arise, models often lack a robust mechanism to reconcile the divergent information, frequently opting to ignore the tool’s output. As shown in Figure 1, the model fails to arrive at the correct answer specifically because it ignores a valid tool result, we define this phenomenon as “Tool Ignored”. This behavior indicates that current TIR models struggle to accurately discern when to trust or dismiss tool result, leading to redundant reasoning paths and erroneous conclusions.

To overcome these limitations, we propose Adaptive Tool Trust Calibration (ATTC), a novel framework that guides the model to adaptively choose to trust or ignore the tool results based on the confidence score of generated code blocks. When a model calling tool is detected, ATTC will score the code blocks generated by the model based on

a specific confidence scoring formula. If the confidence score is greater than the empirically determined threshold, ATTC will guide the model to trust the tool results, otherwise ATTC will guide the model to rethink. We conduct extensive experiments across multiple open-source TIR models, and the experimental results decisively demonstrate that ATTC effectively reduces the “Tool Ignored” issue, resulting in a performance increase of 4.1% to 7.5%.

Overall, our contributions are as follows:

- We find that the TIR model does not know when to trust the results of the tool and define the “Tool Ignored” issue.
- We propose a novel framework ATTC for tool-integrated reasoning that guides the model to adaptively choose to trust or ignore the tool results based on the confidence score of generated code blocks.
- Extensive experiments conducted on open-source TIR models of various sizes demonstrate that ATTC improves the performance of the models.

2 Phenomenon Analysis

2.1 Contradictions of Reasoning and Tools

After careful observation of many trajectories of Tool-Integrated Reasoning, we observe that the conclusions produced by the model’s reasoning are not always consistent with the outputs returned by external tools, in fact conflicts between the two arise frequently. To determine the exact proportion of such conflicting instances and to characterize their resolution, we conduct an LLM-based audit of more than 32k cases followed by a detailed quantitative evaluation. Specifically, we measure the prevalence of conflicts separately in true and false cases. In cases of conflict, we further analyze whether the model tends to rely on its own reasoning or defer to tool outputs. The prompts used to guide this analysis are provided in the Appendix B.

Figure 2 shows that a significant proportion, between 40% and 60%, of the false cases display a contradiction between the model’s reasoning and the output from the external tool. In over half of the conflicting scenarios, the model exhibits a strong tendency to trust its own reasoning over the results produced by the tool. This pattern reveals a significant limitation in the model’s meta-cognition, specifically its inability to effectively determine when to trust the output returned by the tool compared to its own reasoning results.

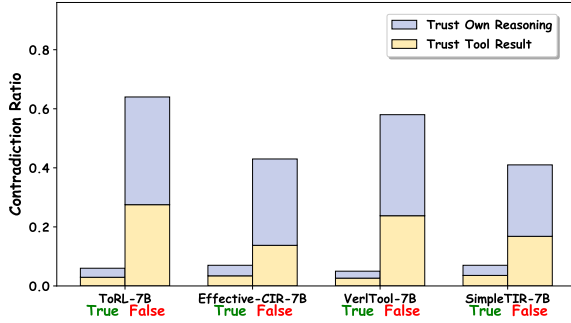


Figure 2: The proportion of contradictions between model reasoning and tool results in true and false cases. The color proportions in the column indicate the proportion that the model chooses to believe in its own reasoning or tool results.

2.2 Tool Ignored

By examining a large set of false cases exhibiting reasoning tool contradictions, we identify a counterintuitive failure mode: when a conflict arises, the tool output is correct, yet the model ignores it and instead adheres to its own reasoning, producing an incorrect answer. We name this failure mode as “Tool Ignored”. This phenomenon indicates a systematic preference for self-generated reasoning over externally provided evidence, which prevents the model from fully leveraging tool augmentation and ultimately degrades task accuracy. A specific case is shown in Figure 1.

To assess the prevalence of this failure mode, we conduct a systematic analysis of false cases across four distinct models and four challenging datasets. The results summarized in Figure 3 show that in every model–dataset combination, “Tool Ignored” accounts for at least 15% of errors. This phenomenon undermines both accuracy and computational efficiency. When a verifiably correct tool output is ignored, the model often tends to generate redundant reasoning steps or tool calls, yielding an incorrect prediction while incurring unnecessary computational cost. It is also important to avoid blindly accepting the results generated by tools. Therefore, the key challenge is to endow the model with meta-cognitive calibration to decide when to trust an external tool and when to rely on its reasoning.

3 Methodology

3.1 Preliminaries

In this work, we consider a Tool-Integrated Reasoning (TIR) setting where a language model inter-

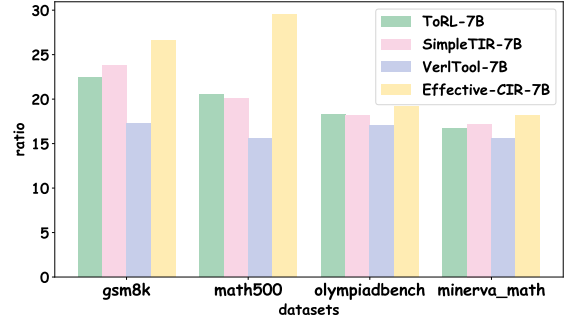


Figure 3: Proportion of “Tool Ignored” phenomenon of different TIR models on four datasets.

acts with an external Python execution environment during test-time reasoning. Under this particular paradigm, code generation is selectively and autonomously triggered by the model during its reasoning process. Crucially, the model is trained to learn how to effectively leverage this generated code to assist, augment, and validate its reasoning capabilities.

Formally, the TIR model maintains a reasoning trajectory $\mathcal{T}^{(t)}$ at iteration t as:

$$\mathcal{T}^{(t)} = \{(r^{(1)}, c^{(1)}, o^{(1)}), \dots, (r^{(t)}, c^{(t)}, o^{(t)})\} \quad (1)$$

where $r^{(t)}$ denotes the natural language reasoning, $c^{(t)}$ represents the generated executable code, and $o^{(t)}$ is the execution result returned by the external environment. The iterative generation process follows:

$$(r^{(t)}, c^{(t)}) \sim M_{\text{tir}}(Q, \mathcal{T}^{(t-1)}) \quad (2)$$

$$o^{(t)} = \mathcal{E}(c^{(t)}) \quad (3)$$

$$\mathcal{T}^{(t)} = \mathcal{T}^{(t-1)} \cup \{(r^{(t)}, c^{(t)}, o^{(t)})\} \quad (4)$$

Given the input prompt Q and the accumulated trajectory $\mathcal{T}^{(t-1)}$, the TIR model M_{tir} continues to generate. The generated code is then executed by an external code execution environment \mathcal{E} to obtain the corresponding output. This iterative process continues until a termination condition is satisfied, at which point the model produces the final answer.

To address the “Tool Ignored” phenomenon discussed in Section 2, we introduce the Adaptive Tool Trust Calibration (ATTC) method to guide the TIR model on whether to trust or ignore the tool in the next section.

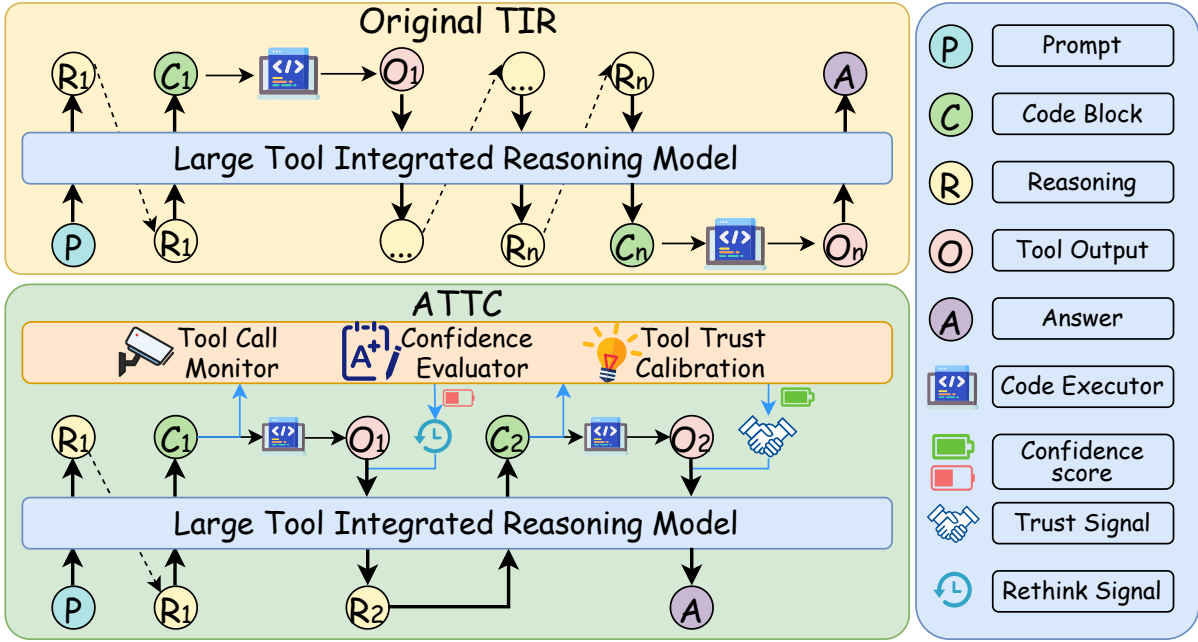


Figure 4: An overview of the Adaptive Tool Trust Calibration (ATTC) method.

3.2 Adaptive Tool Trust Calibration

The main idea behind ATTC is that the model’s confidence in its generated code block indicates how sufficient its prerequisite reasoning is before making a tool call. We observe that code blocks resulting from incomplete or flawed reasoning processes tend to exhibit markedly lower confidence levels. Conversely, when the preceding thought process is comprehensive and logically sound, the model generates code with a significantly higher degree of certainty. Specific quantitative experiments can be found in Section 4.3. This pattern suggests that the TIR model implicitly recognizes the trustworthiness of the tool’s potential output but lacks the explicit mechanism to utilize this awareness in its ongoing reasoning, frequently leading to erroneous trust or ignorance of tool results. ATTC is designed to bridge this gap by converting this implicit awareness into an explicit, actionable control signal.

As shown in Figure 4, ATTC involves three designs to determine whether to trust tool: tool call monitor, confidence evaluator, tool trust recalibrate.

Tool Call Monitor Within the ATTC framework, the tool call monitor module is responsible for actively supervising the model’s generation stream. Due to the generation paradigm of TIR model, we use a rule-based monitor method. Specifically, the tool call monitor detects a tool call and subsequently suspends the model’s generation flow

when dedicated markers, such as “`<tool_result>`” or “`<python> </python>`”, are encountered. Simultaneously, we utilize the module to extract the code block generated by the model, which is usually enclosed within specific delimiters such as “`python`” or “`<python> </python>`”, for use in downstream system processes. The code block for iteration t is as follows: $c^{(t)} = [c_0^{(t)}, c_1^{(t)}, c_2^{(t)}, \dots, c_n^{(t)}]$, where $c_i^{(t)}$ denotes a token in the code block.

Confidence Evaluator The confidence evaluator is designed to compute the confidence of the model-generated code block. Specifically, it defines the confidence of an individual token as the maximum predicted probability assigned to that token by the model. The i -th token $c_i^{(t)}$ in the code block corresponds to the k -th token generated by M_{tir} at iteration t . To derive a single measure for the entire code block, the overall confidence score is then computed as the geometric mean of these token-level confidence scores across all constituent tokens, the confidence \mathcal{C} at iteration t is calculated as follows:

$$p(c_i^{(t)}) = \text{softmax}(H_{tir}(Q, \mathcal{T}^{(t-1)}))_k \quad (5)$$

$$\mathcal{C} = \left(\prod_{i=1}^n \max p(c_i^{(t)}) \right)^{1/n} \quad (6)$$

where H_{tir} is the language model head at the final layer of the TIR model M_{tir} . The calculation of

Model	MATH 500	Minerva Math	Olympiad	AIME24	AMC23	Avg
<i>Models based on Qwen2.5-7B</i>						
Qwen2.5-Math-7B-Instruct	80.2	26.8	41.6	30.0	52.5	46.2
+ATTC	82.8	29.0	50.5	30.0	67.5	52.0 _{+5.8}
ToRL-7B	82.2	33.5	49.9	43.3	65.0	54.8
+ATTC	84.8	43.8	52.4	46.7	72.5	60.0 _{+5.2}
Effective TIR-7B	82.8	30.5	51.9	42.3	70.0	55.5
+ATTC	85.8	42.3	53.5	46.7	77.5	61.2 _{+5.7}
VerlTool-7B	82.0	31.6	49.8	40.0	67.5	54.2
+ATTC	83.4	44.1	50.5	43.3	70.0	58.3 _{+4.1}
SimpleTIR-7B	82.1	30.1	47.4	46.7	75.0	56.3
+ATTC	83.2	46.7	49.2	50.0	77.5	61.3 _{+5.0}
<i>Models based on Qwen2.5-32B</i>						
Qwen2.5-32B-Instruct	81.2	40.1	45.8	20.0	60.0	49.4
+ATTC	83.4	41.5	47.3	30.0	72.5	54.9 _{+5.5}
ReTool-32B	84.6	30.5	60.1	53.3	80.0	61.7
+ATTC	87.4	36.8	62.5	66.7	92.5	69.2 _{+7.5}
SimpleTIR-32B	85.2	33.8	53.8	50.0	80.0	60.6
+ATTC	88.2	36.8	56.9	56.7	85	64.7 _{+4.1}
<i>Models based on Qwen3-4B</i>						
Qwen3-4B-Instruct	55.0	15.4	30.5	20.0	37.5	31.7
+ATTC	56.8	18.0	31.7	26.7	47.5	36.1 _{+4.4}
ReTool-4B	57.0	16.2	27.7	16.7	42.5	32.0
+ATTC	61.8	23.9	32	16.7	52.5	37.4 _{+5.4}
DemyAgent-4B	71.4	17.3	51.9	40.0	75.0	51.1
+ATTC	79.4	22.4	53.5	43.3	77.5	55.2 _{+4.1}

Table 1: Pass@1 performance of the proposed ATTC method across various tool integrated reasoning models on various math benchmarks.

\mathcal{C} utilizes the geometric mean, a choice driven by its sensitivity to low probability values, effectively reflecting the impact of tokens with lower confidence levels. In addition, geometric mean naturally has scale invariance, which can avoid bias caused by sequence length or probability scale changes, and is more suitable as an indicator to measure the overall confidence level of the code block.

Tool Trust Recalibration Finally, the comparison between the confidence score \mathcal{C} and the empirically determined threshold λ governs whether the TIR model should trust the tool result. If $\mathcal{C} \geq \lambda$, the current tool result is deemed trustworthy, and a specific trust control signal is injected. This signal explicitly directs the model to accept the tool’s output and give the answer based on that result. Conversely, if $\mathcal{C} < \lambda$, the model is not instructed to completely disregard the tool output. Instead, a rethink control signal is injected, which explicitly guides the model to utilize the tool result to critically reflect upon its prior reasoning and generated code, thus necessitating a full reconsideration of

the entire inferential process. For detailed prompts, please refer to Appendix D.

4 Experiments

4.1 Experimental Setup

Benchmarks and Metrics To comprehensively evaluate the tool integrated reasoning capabilities of LLMs, We evaluate on multiple mathematical benchmarks: MATH-500 (Hendrycks et al., 2021), Minerva (Lewkowycz et al., 2022), Olympiad (He et al., 2024), AIME24 and AMC23. The primary evaluation metric is Accuracy, which measures the proportion of correct final answers, calculated as the average pass@1 score. In addition, we employ two auxiliary metrics, Token Count and Time Use, representing the average number of tokens generated per sample and the average inference time per sample, respectively. They are used as indicators of the efficiency of tool-integrated reasoning.

Models We apply our method to a series of open-source TIR models trained through reinforcement

Model	MATH 500		Minerva		Olympiad		AIME24		AMC23		Avg	
	Tok↓	Time↓	Tok↓	Time↓	Tok↓	Time↓	Tok↓	Time↓	Tok↓	Time↓	Tok↓	Time↓
<i>Models based on Qwen2.5-7B</i>												
EffectiveTIR-7B	1195	241	838	120	1203	318	1638	129	1503	78	1275	177
+ATTC	833	177	901	119	1283	316	1636	126	1398	71	1210 _{-5%}	161 _{-9%}
SimpleTIR-7B	1538	389	2683	373	2774	624	4444	278	2737	241	2835	381
+ATTC	1316	323	2534	328	2898	653	3723	206	2428	165	2579 _{-9%}	335 _{-12%}
<i>Models based on Qwen2.5-32B</i>												
ReTool-32B	1476	469	1927	422	2316	804	3080	265	2126	200	2185	432
+ATTC	1429	406	1800	360	2042	650	2705	259	1878	184	1970 _{-10%}	371 _{-14%}
SimpleTIR-32B	1377	425	1849	368	2430	636	2865	281	1770	201	2058	382
+ATTC	1204	393	1472	320	2124	666	2865	236	1520	186	1837 _{-11%}	360 _{-6%}

Table 2: Efficiency performance of the proposed ATTC method across various tool integrated reasoning models on various math benchmarks. "Tok" denotes the average token count for a single question. "Time" denotes the average single reasoning time of each benchmark, and the unit is seconds. ↓ indicates that lower values are better.

learning. Some of them are based on Qwen2.5-7B, such as ToRL-7B (Li et al., 2025), VerlTool-7B (Jiang et al., 2025), Effective TIR-7B (Bai et al., 2025) and SimpleTIR-7B (Xue et al., 2025). There are also some based on Qwen2.5-32B and Qwen3-4B, such as ReTool-32B (Feng et al., 2025), SimpleTIR-32B, DemyAgent-4B (Yu et al., 2025b) and so on. We also conducted experiments on the TIR mode of the base models.

Implementation Details All experiments are conducted using the vLLM framework. For the decoding strategy, we align the temperature and top-p settings with the optimal settings provided in the corresponding model’s paper. Please refer to the Appendix C for specific settings. We conduct experiments with 3 random seeds and report the average pass@1 results.

4.2 Experimental Results

Effectiveness Experiment The results in Table 1 show that ATTC consistently outperforms the vanilla tool-integrated reasoning across three model sizes and five benchmark datasets. As evidenced by the experimental results, the performance enhancements delivered by ATTC are comprehensive and consistent. ATTC enhances model performance by an average of 4.1% to 7.5% across five different datasets. Across five benchmarks of varying difficulty, ATTC consistently demonstrates significant performance improvements, indicating that its effectiveness is independent of dataset complexity. As detailed in Section 2, TIR models systematically exhibit the “Tool Ignored” failure mode and often lack calibrated criteria for when to trust external

tools. ATTC addresses this limitation by instructing the model on whether to trust tool outputs or the model’s reasoning, thereby improving accuracy across diverse tasks. ATTC clearly offers advantages across three different model scales, showing its effectiveness regardless of model size. Its benefits are not dependent on the model’s reasoning capabilities or internal representations. Instead, they stem from optimizing the interaction between the model and the tool, as well as improving the utilization of the tool. These results indicate that ATTC serves as a general, training-independent framework for effective tool use.

Efficiency Experiment The results presented in Table 2 demonstrate the token and time consumption of the ATTC approach across various TIR model sizes on five distinct datasets. Overall, ATTC successfully reduces both token and time consumption in the vast majority of scenarios. Although a few isolated cases show negligible increases, ATTC leads to significant average efficiency gains. Specifically, it results in an average reduction of 5% to 9% in token consumption and 9% to 28% in time consumption per model. This indicates that ATTC substantially enhances the operational efficiency of TIR models across various scales and tasks.

4.3 Analysis and Discussion

Robustness of Threshold λ Figure 5 illustrates the average performance of ATTC on five datasets under varying settings of the threshold λ . The results indicate that when λ is set too low, the model completely trusts the tool, leading to a significant

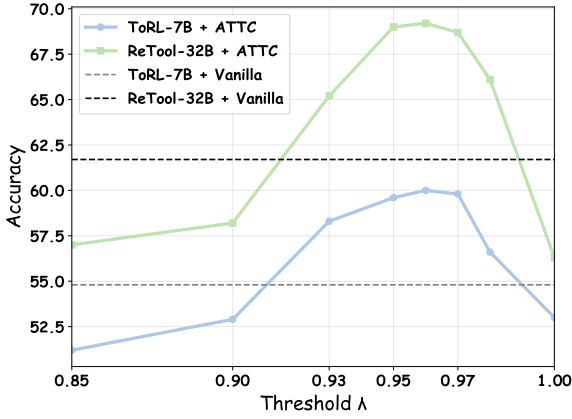


Figure 5: Average performance of ATTC on five datasets under varying settings of the threshold λ .



Figure 6: The linear relationship between PRM score and confidence score.

393 decrease in overall accuracy. Conversely, setting
 394 λ too high causes the model to ignore the tool en-
 395 tirely, also resulting in diminished performance.
 396 Importantly, our method exhibits robustness to λ
 397 within the range of 0.93 to 0.98, suggesting that
 398 fine-grained hyperparameter tuning is not strictly
 399 required. The experimental results presented in
 400 Table 1 uniformly utilized $\lambda = 0.96$. The consis-
 401 tently strong performance across these diverse ex-
 402 periments further validates the generalization abil-
 403 ity and robustness of the ATTC approach.

404 Code Confidence and Reasoning Sufficiency

405 To investigate the relationship between a model’s
 406 confidence scores assigned to self-generated code
 407 blocks and its preceding reasoning process, we em-
 408 ploy a process reward model (PRM) to evaluate
 409 the sufficiency and rationality of the intermediate
 410 reasoning. Specifically, we adopt Universal-PRM-
 411 7B (Tan et al., 2025) as the PRM. As shown in
 412 Figure 6, the PRM scores of the model’s preceding
 413 reasoning are largely proportional to the confidence
 414 scores of the corresponding code blocks. This ob-

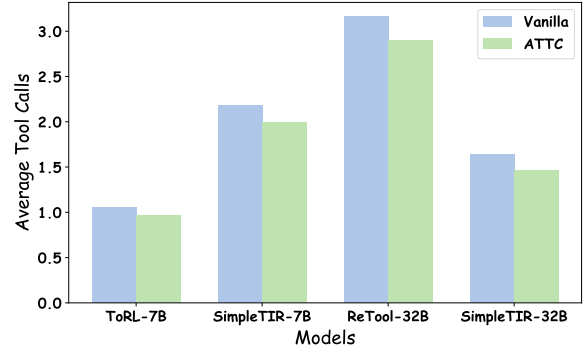


Figure 7: Average number of tool calls per question for four models under the Vanilla and ATTC settings.

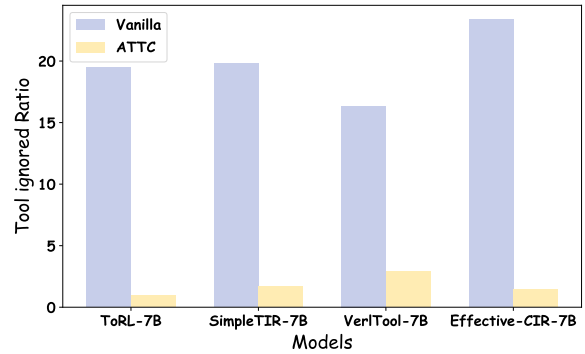


Figure 8: The proportion of ‘‘Tool Ignored’’ among false cases for four models across five datasets under the Vanilla and ATTC settings.

415 servation indicates that the confidence scores as-
 416 sociated with code blocks can effectively reflect
 417 the sufficiency and rationality of the model’s prior
 418 reasoning process.

419 **Tool Call Turns** Figure 7 illustrates the average
 420 number of tool calls per question for four models
 421 under the Vanilla and ATTC settings. We observe
 422 that ATTC does not substantially reduce the overall
 423 number of tool calls, its effect is more pronounced
 424 for models that tend to perform multiple rounds
 425 of tool usage. This behavior aligns with ATTC’s
 426 design motivation, which focuses on eliminating
 427 unnecessary and redundant tool calls while main-
 428 taining essential ones rather than aggressively min-
 429 imizing all tool calls.

430 **Tool Ignored Mitigation** Figure 8 shows the pro-
 431 portion of ‘‘Tool Ignored’’ among false cases for
 432 four models across five datasets under the Vanilla
 433 and ATTC settings. We observe that ATTC substan-
 434 tially alleviates the ‘‘Tool Ignored’’ phenomenon,
 435 demonstrating the effectiveness of the proposed
 436 method. The specific case of ATTC alleviating

Question

The smallest distance between the origin and a point on the graph of $y = \frac{1}{2}x^2 - 9$ can be expressed as a . Find a^2 .

ATTC's response

The screenshot shows the ATTC response for the question. It is divided into several sections:

- Reasoning:** The model states, "To find the smallest distance between the origin and a point (D^2) is 18. So, $(a^2 = 18)$. Let's verify this with Python code to ensure the result is accurate."
- Code Block:** A Python code snippet using SymPy:

```
python
import sympy as sp
# Define the variable
x = sp.symbols('x')
<omitted>
print(min_value)
python
```
- Confidence Score:** A score of 0.974 is shown next to a blue 'A+' icon.
- Tool Call:** The model calls a tool, resulting in an output of `17.000000000000000`.
- Reasoning and Arrive at Final Answer:** The model reflects on its reasoning: "I have great confidence in the code above, I should believe the code execution result. It appears that I made an error in my manual calculations. (D^2) is 17, and thus $(a^2 = 17)$. Therefore, the final answer is: $\boxed{17}$ ".

Figure 9: Case of the same problem optimized by ATTC. The model arrive at an incorrect answer of 18 through reasoning, while the tool provide the correct result of 17. The model trust the tool and give an correct answer.

“Tool Ignored” is shown in Figure 9. Although ATTC greatly decreases the number of issues, there are still some cases where “Tool Ignored” appears. This is due to the inherent tendency of large language models to hallucinate, which makes it difficult for them to completely follow the guidance provided by ATTC. Addressing this limitation may require further investigation in future research.

5 Related work

5.1 Tool Integrated Reasoning

Tool-Integrated reasoning (TIR) (Gou et al., 2024; Wang et al., 2023; Liao et al., 2024) is an advanced paradigm that seamlessly incorporates the call and execution of external tools into the reasoning process. By leveraging external tools such as code executors and search engines, TIR enables LLMs to transcend the inherent limitations of purely reasoning (Lin and Xu, 2025). Early works in TIR primarily relied on prompt engineering (Wang et al., 2025; Yuan et al., 2024; Qian et al., 2024; Chen et al., 2023; Yang et al., 2024b) to guide LLMs in tool call. However, these approaches were heavily rely on meticulously crafted prompts, which limited their scalability and generalizability. Some later works used supervised finetuning (Chen et al., 2025; Qian et al., 2025; Gou et al., 2024; Liao et al., 2024; Schick et al., 2023) to solidify the behavior

pattern of the model actively calling tools in reasoning. Qwen2.5-Math-Instruct (Yang et al., 2024a), ReAct (Yao et al., 2023) and MathCoder (Wang et al., 2023) advances this paradigm by training models on specialized datasets enriched with tool call demonstrations. Through finetuning, the model transitions from passive response to proactive tool call during the reasoning process, where it can autonomously trigger external tools and seamlessly integrate the returned execution results to sustain the reasoning trajectory.

5.2 RL for Tool Integrated Reasoning

SFT-based methods exhibit inherent limitations, as they constrain models to strictly adhere to the tool usage patterns present in the training data distribution, these models often fail to develop adaptive strategies based on the difficulty of the task. Several recent works focus on applying reinforcement learning (Shao et al., 2024; DeepSeek-AI et al., 2025b; Yu et al., 2025a; Liu et al., 2025; Zeng et al., 2025) to address these issues. ReTool (Feng et al., 2025) proposes an automated RL paradigm that allows policy rollouts with multi-turn code execution. ToRL (Li et al., 2025) enables models to discover optimal strategies for tool utilization via unrestricted exploration. VerlTool (Jiang et al., 2025) introduces a unified and modular framework to address multiturn tool interactions. Effective CIR (Bai et al., 2025) develops enhanced training strategies that balance exploration and stability, progressively building tool-use capabilities while improving reasoning performance. SimpleTIR (Xue et al., 2025) stabilizes multi-turn TIR training by filtering out trajectories containing turns that yield neither a code block nor a final answer.

6 Conclusion

In this work, we find some shortcomings in the existing open-source TIR models. There is a common contradiction between the reasoning of model and the results of external tools, and the model tends to believe in its own reasoning. We also discover and define “Tool Ignored”, which represents the situation where the correct results of the tool are ignored by the model, resulting in incorrect answers. We propose a new framework ATTC, and the experimental results from various TIR models and across multiple datasets demonstrate that ATTC effectively reduces the “Tool Ignored” issue, resulting in a performance increase of 4.1% to 7.5%.

513
514
515
516

517
518
519
520
521
522
523
524
525
526
527
528
529

530

531
532
533
534
535

536
537
538
539
540

541
542
543
544
545

546
547
548
549
550
551
552

553
554
555
556
557
558
559
560

561
562
563

Limitations

Although ATTC has generally improved the performance of TIR models, we identify some possible limitations as follows:

- Due to limitations in computing resources, we only conducted experiments on open-source TIR models with a size less than 32B.
- This work focuses exclusively on the application of TIR in scenarios where code executors are utilized as the tool. We have not yet extended our method to scenarios where search engines are utilized as the tool, which we leave for future exploration.
- Currently, optimization is only being conducted during the reasoning stage. In the future, we will consider addressing the problem at its root during training.

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.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2025. [Advancing tool-augmented large language models: Integrating insights from errors in inference trees](#). *Preprint*, arXiv:2406.07115.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. [Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks](#). *Preprint*, arXiv:2211.12588.

Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. [Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities](#). *Preprint*, arXiv:2507.06261.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 179 others. 2025a. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). *ArXiv*, abs/2501.12948.

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. 2025b. [Deepseek-r1: 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](#). *Preprint*, arXiv:2309.17452.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. [Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems](#). *Preprint*, arXiv:2402.14008.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring mathematical problem solving with the math dataset](#). *Preprint*, arXiv:2103.03874.

Yixin Ji, Juntao Li, Yang Xiang, Hai Ye, Kaixin Wu, Kai Yao, Jia Xu, Linjian Mo, and Min Zhang. 2025. [A survey of test-time compute: From intuitive inference to deliberate reasoning](#). *Preprint*, arXiv:2501.02497.

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, Tianyu Pang, and Wenhu Chen. 2025. [Verltool: Towards holistic agentic reinforcement learning with tool use](#). *Preprint*, arXiv:2509.01055.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Thea Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. [Solving quantitative reasoning problems with language models](#). *Preprint*, arXiv:2206.14858.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. [Torl: Scaling tool-integrated rl](#). *Preprint*, arXiv:2503.23383.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. [Mario: Math reasoning with code interpreter output – a reproducible pipeline](#). *Preprint*, arXiv:2401.08190.

Heng Lin and Zhongwen Xu. 2025. [Understanding tool-integrated reasoning](#). *Preprint*, arXiv:2508.19201.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. [Understanding r1-zero-like training: A critical perspective](#). *Preprint*, arXiv:2503.20783.

620	OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer,	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,	677
621	Adam Richardson, Ahmed El-Kishky, Aiden Low,	Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao,	678
622	Alec Helyar, Aleksander Madry, Alex Beutel, Alex	Chengen Huang, Chenxu Lv, Chujie Zheng, Day-	679
623	Carney, Alex Iftimie, Alex Karpenko, Alex Tachard	iheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao	680
624	Passos, Alexander Neitz, Alexander Prokofiev,	Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41	681
625	Alexander Wei, Allison Tam, and 244 others. 2024.	others. 2025. Qwen3 technical report . <i>Preprint</i> ,	682
626	Openai o1 system card . <i>Preprint</i> , arXiv:2412.16720.	arXiv:2505.09388.	683
627	Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi	An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao,	684
628	Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur,	Bowen Yu, Chengpeng Li, Dayiheng Liu, Jian-	685
629	and Heng Ji. 2025. Smart: Self-aware agent for tool	hong Tu, Jingren Zhou, Junyang Lin, Keming Lu,	686
630	overuse mitigation . <i>Preprint</i> , arXiv:2502.11435.	Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang	687
631	Cheng Qian, Shihao Liang, Yujia Qin, Yining Ye, Xin	Ren, and Zhenru Zhang. 2024a. Qwen2.5-math tech-	688
632	Cong, Yankai Lin, Yesai Wu, Zhiyuan Liu, and	nical report: Toward mathematical expert model via	689
633	Maosong Sun. 2024. Investigate-consolidate-exploit:	self-improvement . <i>Preprint</i> , arXiv:2409.12122.	690
634	A general strategy for inter-task agent self-evolution.	Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao,	691
635	<i>Preprint</i> , arXiv:2401.13996.	Minkai Xu, Wentao Zhang, Joseph E. Gonzalez,	692
636	Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta	and Bin Cui. 2024b. Buffer of thoughts: Thought-	693
637	Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola	augmented reasoning with large language models.	694
638	Cancedda, and Thomas Scialom. 2023. Toolformer:	<i>Preprint</i> , arXiv:2406.04271.	695
639	Language models can teach themselves to use tools.	Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak	696
640	<i>Preprint</i> , arXiv:2302.04761.	Shafran, Karthik Narasimhan, and Yuan Cao. 2023.	697
641	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu,	React: Synergizing reasoning and acting in language	698
642	Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan	models . <i>Preprint</i> , arXiv:2210.03629.	699
643	Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024.	Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan,	700
644	Deepseekmath: Pushing the limits of mathemati-	Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan,	701
645	cal reasoning in open language models . <i>Preprint</i> ,	Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin,	702
646	arXiv:2402.03300.	Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan	703
647	Xiaoyu Tan, Tianchu Yao, Chao Qu, Bin Li, Minghao	Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and	704
648	Yang, Dakuan Lu, Haozhe Wang, Xihe Qiu, Wei Chu,	16 others. 2025a. Dapo: An open-source llm re-	705
649	Yinghui Xu, and Yuan Qi. 2025. Aurora:automated	inforcement learning system at scale . <i>Preprint</i> ,	706
650	training framework of universal process reward mod-	arXiv:2503.14476.	707
651	els via ensemble prompting and reverse verification.	Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan,	708
652	<i>Preprint</i> , arXiv:2502.11520.	and Mengdi Wang. 2025b. Demystifying rein-	709
653	Kimi Team, Angang Du, Bofei Gao, Bofei Xing,	forcement learning in agentic reasoning . <i>Preprint</i> ,	710
654	Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun	arXiv:2510.11701.	711
655	Xiao, Chenzhuang Du, Chonghua Liao, Chuning	Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung,	712
656	Tang, Congcong Wang, Dehao Zhang, Enming Yuan,	Hao Peng, and Heng Ji. 2024. Craft: Customiz-	713
657	Enzhe Lu, Feng Tang, Flood Sung, Guangda Wei,	ing llms by creating and retrieving from specialized	714
658	Guokun Lai, and 75 others. 2025. Kimi k1.5:	toolsets . <i>Preprint</i> , arXiv:2309.17428.	715
659	Scaling reinforcement learning with llms . <i>ArXiv</i> ,	Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai	716
660	abs/2501.12599.	Wang, Yang Yue, Shiji Song, and Gao Huang. 2025.	717
661	Hongru Wang, Boyang Xue, Baohang Zhou, Tianhua	Does reinforcement learning really incentivize rea-	718
662	Zhang, Cunxiang Wang, Huimin Wang, Guanhua	soning capacity in llms beyond the base model?	719
663	Chen, and Kam fai Wong. 2025. Self-dc: When to	<i>Preprint</i> , arXiv:2504.13837.	720
664	reason and when to act? self divide-and-conquer	Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Ke-	721
665	for compositional unknown questions . <i>Preprint</i> ,	qing He, Zejun Ma, and Junxian He. 2025. Simplerl-	722
666	arXiv:2402.13514.	zoo: Investigating and taming zero reinforcement	723
667	Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun	learning for open base models in the wild . <i>Preprint</i> ,	724
668	Luo, Weikang Shi, Renrui Zhang, Linqi Song,	arXiv:2503.18892.	725
669	Mingjie Zhan, and Hongsheng Li. 2023. Mathcoder:	Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang,	726
670	Seamless code integration in llms for enhanced math-	Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen	727
671	ematical reasoning . <i>Preprint</i> , arXiv:2310.03731.	Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen	728
672	Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru	Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang,	729
673	Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025.	Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and	730
674	Simpletir: End-to-end reinforcement learning for	3 others. 2025. A survey of large language models.	731
675	multi-turn tool-integrated reasoning . <i>Preprint</i> ,	<i>Preprint</i> , arXiv:2303.18223.	732
676	arXiv:2509.02479.		

A The Use of Large Language Models

We use large language models to assist in analyzing the reasoning trajectories of a large number of TIR models. A large language model was used in writing this manuscript to assist with proofreading and improve the clarity of the text. All knowledge content, including ideas, analysis, and conclusions, is the author’s work.

B Prompt Template

The prompt template used to detect the conflict rate between reasoning and tools is shown in the Figure 10. The prompt template for detecting “Tool Ignored” is shown in the Figure 11. The specific prompt template for different TIR models are shown in Figures 12 to 17.

C Experimental Set

The specific temperature and top_p settings for different TIR models are shown in the table 3

Model	Temperature	Top_p
Qwen2.5-Math-7B-Instruct	0	1
ToRL-7B	0	1
Effective TIR-7B	0.6	0.95
VerlTool-7B	0	1
SimpleTIR-7B&32B	1.0	0.7
Qwen2.5-32B-Instruct	0	1
ReTool-32B&4B	1.0	0.7
Qwen3-4B-Instruct	0	1
DemyAgent-4B	1.0	0.6

Table 3: Temperature and top_p of different TIR models.

D Method Details

During the Tool Trust Recalibration process, when $\mathcal{C} \geq \lambda$, ATTC guides TIR to trust the tool’s results and continue reasoning by inserting special prompts into the original reasoning trajectory. On the contrary, when $\mathcal{C} < \lambda$, ATTC guides the TIR model to consider rewriting the code or rethinking the reasoning path based on the tool results by inserting additional prompts. The specific prompt is shown in the Table 4.

Control Signal	Prompt
Trust	I have great confidence in the code above, i should believe the code execution result.
Rethink	I have no confidence in the code above, i should consider optimizing it or reflecting on whether to change my reasoning path.

Table 4: Method Details.

E Case Studies

Figure 18 shows a case of the comparison between the Vanilla TIR response and ATTC’s response. In vanilla TIR, the model ignores the correct answer provided by the tool and answers incorrectly. In contrast, the ATTC framework guides the model to trust the tool’s results and provide accurate answers.

Prompt Template For Detecting The Contradiction Ratio Between Reasoning And Tools

I will give you a jsonl formatted data, which is a case where the model answered. "question" is the question answered by the model, "answer" is the standard answer, "solution" is the standard problem-solving process, "code" is the content of the model's answer, and "pred" is the final answer of the model.

You need to analyze the model's answer content based on this information.

If there is a contradiction between the inference of the model and the execution result of Python code (after "```output"), the model chooses to believe in the inference, output "\\boxed{0}".

If there is a contradiction between the inference of the model and the execution result of Python code(after "```output"), the model chooses to believe in the code execution result,output "\\boxed{1}".If neither of these situations exist, output "\\boxed{2}".Your answer is limited to these three types and must be included in "\\boxed{}", don't output extra text".

The cases that need to be analyzed are as follows:\n

Figure 10: Contradiction Ratio Prompt Template

Prompt Template For Detecting "Tool Ignored"

I will give you a jsonl formatted data, which is a case where the model answered. "question" is the question answered by the model, "answer" is the standard answer, "solution" is the standard problem-solving process, "code" is the content of the model's answer, and "pred" is the final answer of the model.

You need to analyze the model's answer content based on this information.

If there is no Python code segment(after "```python") in the model's answer, output "\\boxed{2}". If the return result of the Python code (after "```output") is correct (including correct answer and correct thinking), output "\\boxed{1}". Otherwise, output "\\boxed{0}". Your answer is limited to these three types and must be included in "\\boxed{}", don't output extra text".

The cases that need to be analyzed are as follows:\n

Figure 11: "Tool Ignored" Prompt Template

Prompt Template For ToRL-7B&VerlTool-7B

A conversation between User and Assistant. The user asks a question, and the Assistant solves it.

User:Please integrate natural language reasoning with programs to solve the problem above, and put your final answer within \boxed{.

{Question}

Assistant:

Figure 12: ToRL-7B&VerlTool-7B's Prompt Template

Prompt Template For Effective TIR-7B

Please solve the following problem step by step. During your reasoning process, if needed, you can choose to write python code to enhance your reasoning. The code executor will run your code and provide the execution results back to you to support your reasoning process.
Please put the final answer within `\boxed{}`.
Question:
{Question}

Figure 13: Effective TIR-7B 's Prompt Template

Prompt Template For Qwen2.5-Math-7B-Instruct&Qwen2.5-32B-Instruct fonttitle

```
<|im_start|>system
You are Qwen, created by Alibaba Cloud. You are a helpful assistant.
<|im_end|>
<|im_start|>user
Please integrate natural language reasoning with programs to solve the problem
above, and put your final answer within \boxed{}.
{example['question']}
<|im_end|>
<|im_start|>assistant
```

Figure 14: Qwen2.5-Math-7B-Instruct&Qwen2.5-32B-Instruct's Prompt Template

Prompt Template For SimpleTIR-7B&32B

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 `\n```python \n```\n`. You need to use ``print()`` to output intermediate results. The code execution result is only auxiliary, you can choose to believe or not believe it.

Answer Format:

You should use `\boxed{}` to return your final answer. The last part of your response should be: `\boxed{'The final answer goes here.'}`

User Question:

{Question}

Figure 15: SimpleTIR-7B&32B 's Prompt Template

Prompt Template For ReTool-32B&4B

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 `<interpreter>output</interpreter>`) 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.

Code Format:

Each code snippet is wrapped with

```
<code>
```python
code snippet
```
</code>
```

Answer Format:

The last part of your response should be:

```
<answer>\boxed{'The final answer goes here.'}</answer>
```

User Question:

{Question}

Assistant:

Figure 16: ReTool-32B&4B's Prompt Template

Prompt Template For DemyAgent-4B&Qwen3-4B

Analyze and solve the following [math/science domain] problem step by step.

Problem: {Question}

Hint: The tool could be used for more precise and efficient calculations and could help you to verify your result before you reach the final answer.

Note: You should first analyze the problem and form a high-level solution strategy, then utilize the tools to help you solve the problem.

Answer Format: Do not put units of the final answer inside `\boxed{}`. The content of `\boxed{}` should be the numerical value of the final answer only, without any units. Remember once you make sure the current answer is your final answer, do not call the tools again and directly output the final answer in the following text format, the answer format must be: `\boxed{'The final answer goes here.'}`.

Figure 17: DemyAgent-4B&Qwen3-4B's Prompt Template

Question

How many sides would there be in a convex polygon if the sum of all but one of its interior angles is 1070°?

Vanilla TIR's response

Reasoning

To determine the number of sides of a convex polygon given that the sum of all but one of its interior angles is (1070°) , we need to use the properties of the interior angles of a polygon. **Let's solve this step by step using code to iterate over possible values of (n) and find the correct one.**

Code Block

```
python
import math

# Function to find the number of sides of the polygon
def find_number_of_sides():
    for n in range(3, 1000): # Starting from 3 because a polygon has at least 3 sides
        sum_of_interior_angles = (n - 2) * 180
        # We need to find x such that (n-2) * 180 - x = 1070
        # So x = (n-2) * 180 - 1070
        x = (n - 2) * 180 - 1070
        # x must be between 0 and 180
        if 0 < x < 180:
            return n

# Find the number of sides
number_of_sides = find_number_of_sides()
print(f"The number of sides of the polygon is: {number_of_sides}")
```

Tool Call

```
output
The number of sides of the polygon is: 8
```

Reasoning and Arrive at Final Answer

It appears there was an error in the initial reasoning or the code logic. Let's recheck the problem carefully. Since $(0^\circ < 10^\circ < 180^\circ)$, this value is valid. Therefore, the number of sides of the polygon is indeed (12) . So the correct answer is: (12)

ATTC's response

Reasoning

To determine the number of sides (n) of a convex polygon given that the sum of all but one of its interior angles is (1070°) , we need to use the properties of the interior angles of a polygon. **Let's write the Python code to verify this solution step-by-step:**

Code Block

```
python
import sympy as sp

# Function to calculate the number of sides of the polygon
def find_number_of_sides(sum_of_angles):
    for n in range(3, 100): # We start from 3 since a polygon has at least 3 sides
        total_sum_of_angles = 180 * (n - 2)
        remaining_angle = total_sum_of_angles - sum_of_angles
        if remaining_angle < 180 and remaining_angle > 0:
            return n
    return None

# Given sum of all but one of the interior angles is 1070 degrees
sum_of_angles = 1070

# Find the number of sides
number_of_sides = find_number_of_sides(sum_of_angles)
print(number_of_sides)
```

Confidence Score

A+ 0.968

Tool Call

```
output
8
```

Reasoning and Arrive at Final Answer Tool Trust Recalibrate

I have great confidence in the code above, I should believe the code execution result. Thus, the number of sides of the convex polygon is (8) . The final answer is (8) .

Figure 18: A case of the comparison between the Vanilla TIR response and ATTC's response.