Towards Better Function Calling in LLMs With a Path-Aware Reinforcement Learning Framework

Anonymous ACL submission

Abstract

Large language models (LLMs) show great promise in solving complex tasks through external tool use. However, existing approaches largely focus on standard tool formats and instruction-following, neglecting the broader problem of generalizable and robust tool interaction. In this paper, we explore three underexamined challenges in LLMs' tool-use capabilities: adaptation to counterintuitive tool rules, autonomous discovery of tool functionality under incomplete specifications, and the impact of historical memory on tool-use efficiency. To address these challenges, we propose Path-Aware Reinforcement Learning (PARL), a novel framework that integrates trajectory-level reward assignment and history-based contextualization. PARL assigns dynamic rewards based on path-level outcomes and relative tool-use efficiency, while maintaining a fixed-size memory window to guide policy learning. Experiments across diverse non-standard tool-use scenarios demonstrate that PARL consistently outperforms existing methods, achieving relative gains of up to 28.9% in low-information settings and 19.8% in counterfactual reasoning. Our work provides both a diagnostic benchmark and an effective reinforcement learning strategy for advancing tool-augmented LLMs.Our code and dataset will be available at XXX

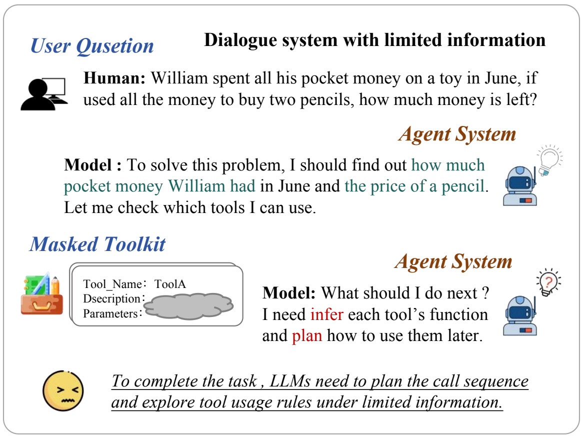

Figure 1: When faced with problems involving unknown information, large language models decompose the task, plan an appropriate tool invocation path, and generate call instructions in a specific format based on tool definitions.

1 Introduction

Large language models (LLMs) have achieved remarkable breakthroughs in processing human language (Achiam et al., 2023; Chang et al., 2024; Kumar et al., 2025), demonstrating exceptional capabilities in text comprehension and generation. However, when solving many real-world problems, LLMs require to invoke external tools (Brown et al., 2020; Lewis et al., 2020; Resnik, 2025; Gu et al., 2025). Establishing a mechanism that enables better interactions between LLMs and tools, can greatly extend the boundaries of model capabilities, transforming them from pure text generators into interactive intelligent agents (Schick et al., 2023; Yao et al., 2023; Shen et al., 2023; Hu et al., 2025). How to better interact with tools is still an open problem. Most existing work improves tool-use efficiency by optimizing tool formats,such as standardizing interfaces and compressing documentation (Qin et al., 2023; Yuan et al., 2024; Ni et al., 2025; Li et al., 2023), which heavily rely on the model's instruction following ability. In addition, post-training methods (Feng et al., 2025; Singh et al., 2025; Kang et al., 2025) like supervised fine-tuning and reinforcement learning have been proposed to enhance invocation accuracy and reasoning capabilities. Although these methods perform well in specific scenarios, they generally rely on strict parameter specifications and fixed input formats, overemphasizing the model's instruction-following ability rather than its deep understanding of tools. More critically, they focus solely on standard usage contexts and lack exploration of non-standard scenarios, limiting our understanding of the upper bound of LLMs' tool-use capabilities.

In this paper, we investigate three fundamental questions about LLMs' tool-use capabilities: (1) Can models adapt to counterintuitive tool rules when such rules are not explicitly provided? (2) To what extent can models autonomously discover tool functionalities without complete specifications? (3) How does maintaining a memory of past interactions affect tool-use performance? We design specialized experiments to address these questions, including scenarios where mathematical operation precedence is reversed and tool descriptions are progressively obfuscated. Based on our findings, we propose Path-Aware Reinforcement Learning (PARL), a novel framework that enhances tool-use capabilities through two key mechanisms: (1) fine-grained reward signals based on intra-group relative advantage and outcome correctness applied to entire tool-invocation paths, and (2) a history-aware training process that maintains a fixed-size memory window of past tool interactions. Experimental results demonstrate that our approach consistently outperforms state-of-the-art methods across all scenarios, achieving average improvements of 19.8% on counterfactual rule understanding, 28.9% on tool discovery with minimal information, and 14.2% on complex tool composition tasks.In summary, the main contributions of our work are:

- We design a comprehensive evaluation framework examining LLMs' tool-use capabilities in challenging scenarios.
- We propose Path-Aware Reinforcement Learning, a novel framework combining path-dependent rewards with historical context to enhance tool-use performance across diverse scenarios.
- We demonstrate significant performance improvements, particularly in adapting to reversed mathematical rules and discovering tool functionality without explicit descriptions.

2 Related Work

2.1 Tool Integration in LLMs

Recent work has integrated external tools with LLMs through various approaches: API invocation frameworks, reasoning-action interleaving, and model orchestration systems(Schick et al., 2023; Yao et al., 2023; Shen et al., 2023). While specialized benchmarks and interface standardization efforts have advanced the field(Li et al., 2023; Huang et al., 2023; Basu et al., 2024), current approaches predominantly assume well-specified tool interfaces and rely heavily on instruction-following capabilities(Basu et al., 2024; Yuan et al., 2024). This fundamental assumption limits adaptability when tools exhibit non-standard behaviors or provide incomplete specifications(Basu et al., 2024; Yang et al., 2024b). Most existing systems struggle with counterfactual usage patterns or minimal documentation(Gao et al., 2024; Luo et al., 2025; Hsieh et al., 2023), highlighting the need for more adaptive tool interaction mechanisms that can operate effectively beyond familiar training distributions.

2.2 Post-Training Methods for Tool Learning

Post-training optimization for tool use primarily follows two approaches: supervised fine-tuning and reinforcement learning(Chen et al., 2024; Xu et al., 2025; Wang et al., 2025). While fine-tuning methods can effectively capture tool-specific patterns from demonstrations(Furuta et al., 2023; Shinn et al., 2023; Qian et al., 2025), they often require extensive examples and struggle with novel compositional tasks. Reinforcement learning offers improved adaptability through techniques like relative advantage computation and reward shaping(Yu et al., 2024; Lu et al., 2023), but current methods typically evaluate only final outcomes and treat each interaction in isolation(Lyu et al., 2024; Shi et al.). These limitations restrict performance in scenarios requiring adaptation to implicit rules or tool discovery under minimal information. Our work addresses these gaps through path-dependent reward mechanisms that connect intermediate decisions to final outcomes, combined with a historical memory system that enables cumulative learning from past interactions.

3 Path-Aware Reinforcement Learning Framework

We formulate tool learning as a sequential decision process where a model generates a tool calling sequence s for a user query q using available tools $\mathcal{T}$. To effectively address this challenge, we adopt Grouped Relative Policy Optimization (GRPO) (Shao et al., 2024)as our foundational framework and design a path-aware reward mechanism.

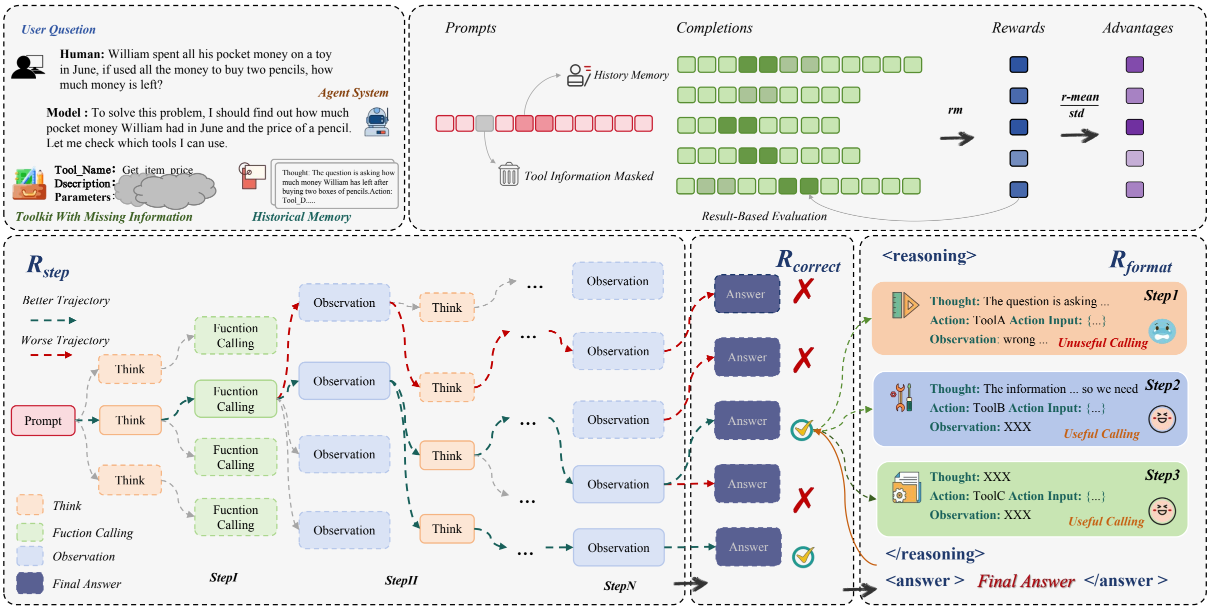

Figure 2: Overview of our Path-Aware Reinforcement Learning framework. The approach consists of three key components: (a) GRPO-based optimization for stable policy updates, (b) path-dependent multi-level rewards that assess both intermediate steps and final outcomes, and (c) historical memory mechanism that maintains context from past interactions.

3.1 Training Objective

The GRPO framework optimizes model parameters θ through the following loss function:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,s)}\Big[\min\Big(\rho_\theta(s|q)A(s,q), \\ \text{clip}(\rho_\theta(s|q), 1-\varepsilon, 1+\varepsilon)A(s,q)\Big)\Big] \quad (1)$$

Here, μ_q and σ_q are the mean and standard deviation of rewards across all samples within the same query group.And our overall reward is constructed from three components:

$$R_{\text{final}}(s,q) = R_{\text{format}}(s) + R_{\text{result}}(s,q) + R_{\text{tool}}(s,q) \quad (2)$$

where $R_{\text{format}}(s)$ evaluates response format compliance, $R_{\text{result}}(s,q)$ measures final answer accuracy, and $R_{\text{tool}}(s,q)$ is our proposed path-aware tool calling reward.

3.2 Path-Aware Reward Mechanism

To evaluate tool usage more holistically, we propose a path-aware reward mechanism that integrates both the efficiency of tool invocation and the correctness of the final outcome. Unlike prior approaches that assign reward per isolated tool call, our design treats the entire interaction trajectory as a functional unit, allowing more accurate credit assignment during reinforcement learning.

We first define the relative invocation ratio of a sample s within a batch (or group) as:

$$r_{\text{call}}(s) = \frac{n(s)}{\bar{n}\text{group}} \quad (3)$$

where $n(s)$ denotes the number of tool calls in the sample s, and $\bar{n}$group is the average tool usage count across the batch.

The efficiency reward is then determined based on the number of tool calls made and the unit reward per call, capped by a maximum efficiency bonus:

$$R_{\text{eff}}(s) = \min\left(n(s) \cdot r_{\text{unit}},\ R_{\text{max}}\right) \quad (4)$$

where r_{unit} is the per-call reward and R_{max} sets an upper limit to prevent over-rewarding excessive but unhelpful invocations.

We then assign the final reward based on the model's output correctness. If the answer is correct, the efficiency reward is kept as is. If the answer is incorrect, the reward is down-weighted:

$$R_{\text{tool}}(s) = \begin{cases} R_{\text{eff}}(s), & \text{correct} \\ \frac{1}{2}R_{\text{eff}}(s), & \text{incorrect} \end{cases} \quad (5)$$

This formulation ensures that correct outputs receive the full efficiency-based reward, while incorrect outputs are still weakly rewarded in proportion to their tool usage, helping the model learn from failure without encouraging wasteful invocation.

Interaction Record Template
Query: [User query] Tool called: [ToolName] Arguments: arguments Observation: [Output returned by the tool] Reasoning: [Model's analysis and decision process] Final answer: [Generated response] Result: [Correct / Incorrect]

Table 1: Template for a single record in the historical memory window.

3.3 Historical Memory Mechanism

Historical interactions provide crucial guidance for the model's tool-invocation decisions. To exploit this, we incorporate a historical memory mechanism into the PARL framework: during training, we maintain a sliding window of up to 500 tokens capturing the model's most recent tool interactions. Before each new decision, these entries are prepended to the model's context, allowing it to draw on past successes and failures. By observing patterns in tool usage, the model learns to adopt more effective invocation strategies and to avoid repeating prior mistakes, even under ambiguous or incomplete tool specifications.

Each record in the historical window follows the structured template shown in Table 1. Unlike conventional experience-replay buffers in reinforcement learning, our approach integrates directly into the language model's context window, naturally serving as an implicit replay buffer while preserving the simplicity of end-to-end training.

4 Experimental Setup

4.1 Dataset

In this section, we introduce two specialized datasets for evaluating tool-calling capabilities: the Calculator Tool Dataset and the Multi-Function Tool Dataset. The former focuses on assessing models' adaptability to different computational rules, while the latter tests their ability to combine multiple tools for solving complex problems. These datasets cover various tool invocation scenarios, from simple single-tool usage to complex tool compositions, featuring two distinct sets of tools (four arithmetic tools and four query tools) designed to systematically evaluate models' performance across different tool-using contexts.

Calculator Tool Dataset The Calculator Tool Dataset evaluates a model's adaptability to tool usage rules, particularly focusing on rule systems that contradict pretrained knowledge. It contains 6,000 mathematical expressions, each comprising up to eight sequential computational steps. To quantitatively measure rule adaptability, we define three rule variants: the standard mathematical precedence rule **(Natural)**, the strict left-to-right computation rule **(Linear)**, and the inverted precedence rule **(Reversal)**. Models are required to execute these computations using four basic arithmetic tools (addition, subtraction, multiplication, and division), with all outputs rounded to two decimal places. Through Python preprocessing, we generate ground-truth answers for all expressions under the three rule systems. For each rule system, we split the 6,000 samples into 5,000 training samples and 1,000 testing samples, yielding a total of 15,000 training samples and 3,000 testing samples.

Multi-Function Tool Dataset The Multi-Function Tool Dataset evaluates a model's ability to discover tool functionalities and compose multiple tools effectively. This dataset features four distinct query tools: get_financial_statement (retrieving financial data),query_date (accessing calendar information), query_item_price (retrieving product prices), and get_relationship_chain (querying relationships between entities), each designed for specific information retrieval tasks. Based on complexity, we divide the dataset into three categories: Single-Tool (requiring 1 tool only), Dual-Tool (requiring 2 tools), and Multi-Tool (requiring 3 or more tools), where models must invoke the appropriate combination of tools to obtain necessary information for answering questions correctly. We constructed this dataset through template-based generation followed by human verification. Each category contains 800 training samples and 200 testing samples, yielding a total of 2,400 training samples and 600 testing samples.

4.2 Baseline Methods

To comprehensively evaluate the effectiveness of our approach, we select the following representative baselines that span various technical approaches from zero-shot to supervised learning to reinforcement learning: (1) **Base Model**: The original large language model without any additional training, evaluated using the same prompts, serving as a performance baseline. (2) **Think_SFT** (Jaech et al., 2024): A supervised fine-tuning method that first extracts thought trajectories and tool invoca-

Rule	Method	SFT/RL	Qwen-1.5B	Qwen-7B	Qwen-think	Llama-3B
Natural	Base	-	61.90	64.50	69.70	43.80
	Think_SFT	SFT	66.40 $_{\uparrow 4.50}$	68.80 $_{\uparrow 4.30}$	71.40 $_{\uparrow 1.70}$	47.90 $_{\uparrow 4.10}$
	GRPO	RL	78.00 $_{\uparrow 16.10}$	78.60 $_{\uparrow 14.10}$	82.20 $_{\uparrow 12.50}$	52.50 $_{\uparrow 8.70}$
	Tool_RL	RL	79.20 $_{\uparrow 17.30}$	81.00 $_{\uparrow 16.50}$	81.30 $_{\uparrow 11.60}$	53.10 $_{\uparrow 9.30}$
	Ours	RL	**82.50** $_{\uparrow 20.60}$	**84.40** $_{\uparrow 19.90}$	**84.20** $_{\uparrow 14.50}$	**55.30** $_{\uparrow 11.50}$
Linear	Base	-	14.70	14.10	15.00	13.70
	Think_SFT	SFT	18.80 $_{\uparrow 4.10}$	20.20 $_{\uparrow 6.10}$	19.40 $_{\uparrow 4.40}$	17.60 $_{\uparrow 3.90}$
	GRPO	RL	48.40 $_{\uparrow 33.70}$	54.50 $_{\uparrow 40.40}$	58.00 $_{\uparrow 43.00}$	33.20 $_{\uparrow 19.50}$
	Tool_RL	RL	51.00 $_{\uparrow 36.30}$	53.80 $_{\uparrow 39.70}$	59.50 $_{\uparrow 44.50}$	35.80 $_{\uparrow 22.10}$
	Ours	RL	**53.50** $_{\uparrow 38.80}$	**57.10** $_{\uparrow 43.00}$	**62.40** $_{\uparrow 47.40}$	**42.00** $_{\uparrow 28.30}$
Reversal	Base	-	1.60	1.50	2.00	1.50
	Think_SFT	SFT	8.80 $_{\uparrow 7.20}$	7.60 $_{\uparrow 6.10}$	10.20 $_{\uparrow 8.20}$	7.80 $_{\uparrow 6.30}$
	GRPO	RL	32.80 $_{\uparrow 31.20}$	26.20 $_{\uparrow 24.70}$	27.10 $_{\uparrow 25.10}$	19.80 $_{\uparrow 18.30}$
	Tool_RL	RL	31.20 $_{\uparrow 29.60}$	28.80 $_{\uparrow 27.30}$	29.50 $_{\uparrow 27.50}$	23.70 $_{\uparrow 22.20}$
	Ours	RL	**33.20** $_{\uparrow 31.60}$	**37.10** $_{\uparrow 35.60}$	**40.60** $_{\uparrow 38.60}$	**26.50** $_{\uparrow 25.00}$

Table 2: Accuracy (%) of different models on the calculator task under three mathematical rule systems. Bold numbers indicate the highest accuracy in each column, and subscripts denote absolute improvement percentages over the Base model.

tion sequences from the high-performing Deepseek-R1 model under identical inputs, then trains the target model using this data containing intermediate reasoning processes. (3) **GRPO** (Guo et al., 2024): The Group Relative Policy Optimization approach proposed by Deepseek, which guides the model to generate correct outputs through a multi-faceted reward mechanism combining format awareness and result correctness. This method represents a mainstream reinforcement learning approach for tool-use training.(4) **ToolRL** (Qian et al., 2025):A reinforcement learning approach that treats tool use as compositional reasoning, featuring fine-grained reward signals for tool name selection, parameter filling, and execution efficiency, with dynamically adjusted reward weights throughout training. This method represents recent advances in the tool-calling domain.(5) **Ours**: Our proposed path-aware reinforcement learning framework as shown in Section(§ 3).

4.3 Evaluation Setup

We use accuracy as our evaluation metric, defined as the match rate between model-generated and ground-truth answers . To assess generalizability across diverse model architectures, we conduct experiments on: Qwen2.5-3B-instruct and Qwen2.5-7B-instruct (Qwen-1.5B, Qwen-7B) (Yang et al., 2024a), Qwen3-1.7B-MOE (Qwen-think) (Yang et al., 2025), and Llama3-8B-instruct (Llama-3B) (Grattafiori et al., 2024). For our methods, we set the batch size to 16 and the generate-num to 8 . Tool invocations were formatted in JSON, with a maximum sequence length of 4096. For more details on the dataset and training, please refer to Appendix A and Appendix B.

5 Experiments

5.1 Exp-I: Can LLMs Discover Latent Rules in Tool Composition?

In this experiment, we investigate whether LLMs can acquire specific tool usage rules—particularly counterfactual through post-training adaptation. We utilize the calculator dataset introduced in Section 4.1 and conduct training and evaluation under three distinct mathematical rule systems. During training, no explicit instructions about tool usage rules are provided; instead, the model must explore different computational strategies and tool compositions through interaction (detailed prompts are provided in Appendix C). As shown in Table 2, all models perform best under the Natural rule system, while performance drops significantly under the Linear and Reversal rule systems, which contradict standard mathematical conventions. For Think_SFT, even with access to ground-truth reasoning paths and tool invocation trajectories, accuracy improvements under counterfactual settings remain limited , suggesting that supervised fine-tuning struggles to override the model's pretrained mathematical knowledge. In contrast, reinforcement learning (RL)-based methods consistently outperform supervised fine-tuning across all rule types.Under the most challenging Reversal rule, our method achieves an average improvement of 32.70% across all model sizes, with particularly

Details	Base	SFT	GRPO	Tool_RL	Ours
Qwen-1.5B					
Full	6.70	11.83	29.17	33.33	**45.17**
TD	1.50	9.17	27.33	31.50	**36.67**
PD	2.67	8.50	25.50	32.83	**33.67**
Hidden	0.17	7.67	23.00	26.00	**29.67**
Qwen-7B					
Full	6.00	14.17	28.16	41.00	**45.00**
TD	5.17	12.50	27.17	28.50	**36.17**
PD	4.33	14.83	29.67	30.83	**36.50**
Hidden	1.83	12.17	25.00	28.50	**33.00**
Qwen-think (MoE)					
Full	26.67	33.67	56.83	71.33	**78.83**
TD	15.00	24.50	37.17	37.50	**58.67**
PD	9.00	28.83	43.50	41.00	**65.17**
Hidden	2.17	16.67	23.83	39.67	**41.67**

Table 3: Accuracy (%) of five training strategies on three model sizes under varying tool-composition complexities.

strong performance on the Qwen3-think model (40.60%). This indicates that RL can effectively reshape the model's internal computation strategies beyond surface-level pattern matching. By leveraging reward signals, RL encourages exploration of high-reward behaviors that may conflict with pretrained biases, facilitating the discovery of rule-compliant tool-use strategies even when they contradict conventional mathematical principles.

5.2 Exp-II: Can LLMs Discover Tool Functions Under Limited Information?

In this section, we investigate whether language models are capable of autonomously discovering tool functionalities and invoking them correctly under limited descriptive information. Tool invocation is regarded as an extension of the model's instruction-following capability, which typically relies on access to complete tool specifications. To systematically evaluate the impact of information constraints on tool-use behavior, we design four progressively weakened settings, where all tool names are replaced with abstract identifiers (`Tool-N`):(1) **FULL**: Full access to tool descriptions, including both functionality and parameter information.(2) **Tool Description (TD)**: Only functional descriptions are provided; parameter information is removed. The model must infer how to use the parameters based on the textual content.(3) **Parameter Description (PD)**: Only parameter format is preserved; functional descriptions and usage strategies are replaced with abstract identifiers, offering minimal semantic guidance.(4)

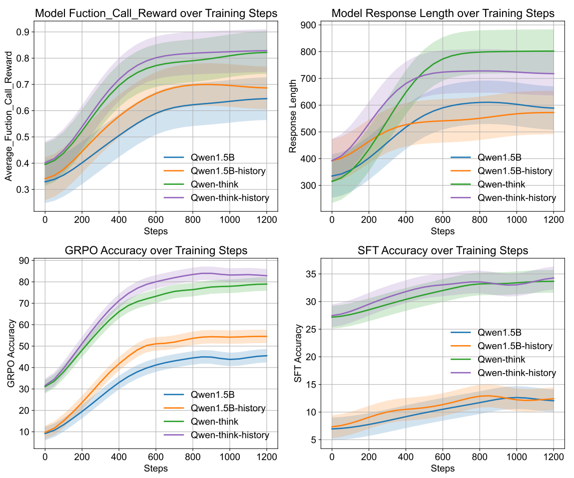

Figure 3: Impact of historical memory on learning efficiency across training steps. The four panels show: (a) Average reward signals, (b) Output sequence length ,(c) RL method accuracy with/without history information and (d) SFT method accuracy with/without history information.

Hidden: All descriptive and semantic information is removed. The model must rely solely on interactive feedback to explore tool functionalities. For supervised fine-tuning experiments, we apply strict string-matching filters to remove all tool-relevant reasoning content, ensuring that models cannot directly learn tool semantics from demonstration data. We train and evaluate on the four query tools introduced in Section 4.1, with results summarized in Table 3.The results show that base models rely heavily on explicit tool information—Qwen-1.5B's accuracy drops from 6.70% (FULL) to 0.17% (Hidden). Similar trends are observed across all model sizes. In contrast, reinforcement learning consistently outperforms supervised fine-tuning under all levels of information constraint, with our method achieving the highest accuracy, especially under the most challenging Hidden setting. These findings suggest that RL not only improves generalization but also enables active exploration, allowing models to infer tool usage strategies from limited feedback rather than relying solely on imitation.

5.3 Exp-III: Can LLMs Learn Effectively from Historical Interactions?

This section investigates the effect of historical interaction memory on the efficiency of learning tool-use strategies, including question formulation, tool invocation, and observation interpretation. In our experimental setup, we maintain a 500-token sliding window that records past tool interactions

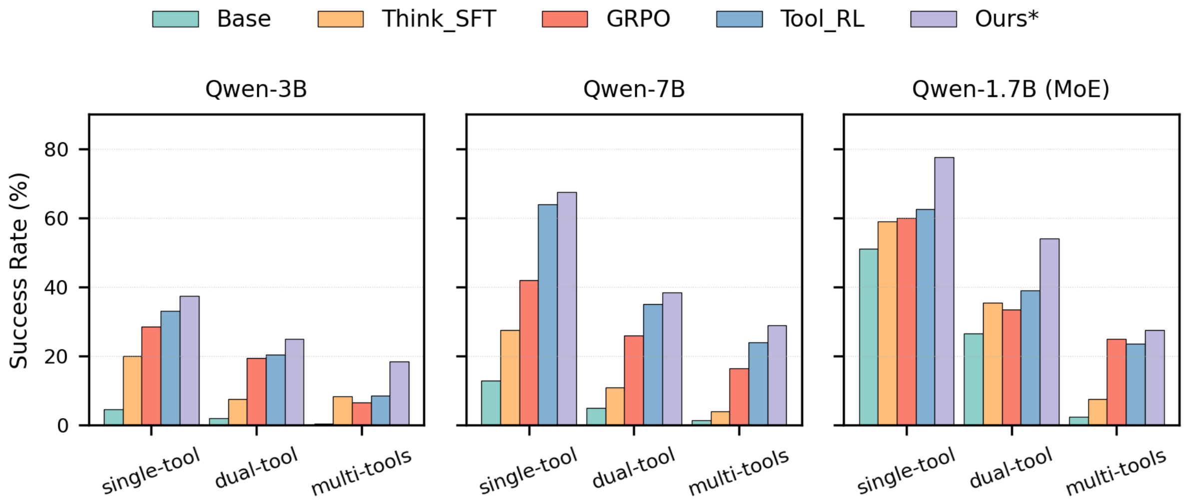

Figure 4: Performance of different models across increasing tool composition complexity in Experiment III. Accuracy declines as composition complexity increases, highlighting the limitations of supervised learning and the robustness of reinforcement learning in multi-tool reasoning tasks.

and relevant context, such as previous tool inputs, outputs, and environmental feedback. To capture the dynamics of learning, we sample multiple training checkpoints across the entire training trajectory. We compare three configurations: Think_SFT , our reinforcement learning method without memory, and our reinforcement learning method augmented with historical memory. The evaluation centers on four key metrics: (a) progression of the average reward signal, (b) output sequence length, and the accuracy of both RL and SFT approaches, denoted as (c) and (d), respectively. Figure 3 presents the temporal evolution of these indicators over training steps.Experimental results demonstrate that integrating historical memory significantly improves learning efficiency in reinforcement learning settings. Models equipped with access to past interactions not only receive stronger reward signals in early training but also achieve faster and more stable convergence in accuracy. These models consistently outperform their history-free counterparts across all metrics. We attribute these improvements to two key advantages: (1) historical records provide informative behavioral patterns that help models identify effective tool-use strategies; and (2) they act as a lightweight, implicit replay buffer, mitigating instability in policy learning and reducing the likelihood of suboptimal convergence. Although supervised fine-tuning also benefits slightly from history, the gains are marginal due to its static and imitative nature, where historical context cannot influence exploration. In contrast, reinforcement learning thrives on dynamic feedback, allowing history to play an active role in guiding policy optimization. Additionally, our history-aware RL models generate shorter, more targeted tool-use sequences while maintaining high correctness, leading to improvements in both reasoning efficiency and computational cost.

Method	Exp1	Exp2	Exp3
Base Model	28.90	2.17	47.10
+ FAR	30.85	18.60	57.25
+ Tool	42.80	32.50	51.40
+ Step	56.40	34.67	53.85
Ours	**62.40**	**41.67**	**68.75**

Table 4: Ablation results comparing different reward strategies under different reward settings.

6 Discussion

6.1 Ablation Study on Reward Design

We conduct ablation studies to evaluate the contribution of individual reward components in our framework. Using qwen-think as our base model, we compare five configurations: 1)**Base**:Base Model without post-training; 2)**FAR**: Format and Result correctness rewards only; 3) **Tool**: Tool usage rewards only; 4) **Step**: Result-conditioned tool step rewards; 5)**Ours**: Full reward system combining all components with path-aware optimization. Experience results are shown in Table 4 .Experimental results show that different reward functions

Question: William spent all his pocket money on a toy in June. If William used all the money to buy two pencils, how much money is left?

 # Model Answer of 200 steps

<reasoning>
Thought : The question is asking William's pocket money and how much money has left after buying pencils.
Action: Tool_A Action Input: {"Personal": "William"}
Observation: Query failed.
Thought : ToolA query failed, so we need to try another approach....
Action: Tool_B Action Input: {"Personal": "William"}
Observation: Query failed.
Thought : ToolB query also failed. We need to try...
Action: Tool_C Action Input: {"Personal": "William"}
Observation: Query failed.
......
</reasoning>
<answer>The information about William's pocket money is not available through the provided tools.
</answer>

Failed

 # Model Answer of 1200 steps

<reasoning>
Thought: The question is asking how much money William has left after buying two boxes of pencils.
Action: Tool_D **Action Input:** {"Personal": "William"}
Observation: William's neighbor is David.
Thought: The information provided by the tool **is not helpful** for solving the question .
Action: Tool_B **Action Input:** {"Item": "pencils"}
Observation: For pencils, the unit price is \$3.5.
Thought: The information provided by the tool is helpful for solving the question.
Action: Tool_A **Action Input:** {"Personal": "William"}
Observation: .....William's pocket money in June was \$654 ......
Thought: William's pocket money in June was \$654. He spent all the money to buy two pencils,which cost \$3.5 × 2 = \$7. So, the money left is \$654 - \$7 = \$647.
</reasoning>
<answer> Final Answer: **\$647** </answer>

Success

Figure 5: Case study comparing qwen-think model behavior after 200 steps (left) versus 1200 steps (right) of RL training. The example shows progressive improvement in: (a) tool selection strategy, (b) error recovery capabilities, and (c) compositional reasoning under tool-obfuscated settings, where no tool descriptions are provided.

all contribute positively to the model's tool-use reasoning, with our method achieving the best overall performance.

6.2 Post-training evaluation of capabilities in tool-composition scenarios.

In the Section (§ 5.2) experiments, we examined the model's ability to adapt to tool usage rules and to autonomously discover tool functionality. This section further investigates the upper bound of tool composition capabilities under full information settings, with a particular focus on the comparative advantages of reinforcement learning over traditional supervised learning in compositional tasks.In this experiment, models are provided with complete tool descriptions and are allowed to use all tools introduced in Experiments I and II. As shown in Figure 4, across all model scales and training strategies, accuracy consistently decreases as the complexity of tool composition increases. This indicates that even with full access to tool information, multi-tool reasoning remains a core bottleneck for current models.

6.3 Case Study

We present a case study based on outputs from the qwen-think model (Figure 5). The model must (i) retrieve William's pocket money and (ii) query the pencil price, without access to tool descriptions or parameter schemas. At an early stage of training, the model frequently calls irrelevant tools, reflecting limited understanding but some exploratory behavior. After 1200 RL steps, it adopts more reliable strategies discarding uninformative results and switching tools efficiently. Despite no explicit tool guidance, it demonstrates improved accuracy and reasoning efficiency, indicating implicit learning of tool functionalities. Appendix D for details.

7 Conclusion

This work investigates the limitations of current LLM tool-use approaches in handling non-standard, under-specified, or counterintuitive scenarios. We introduce three challenge settings that probe models' ability to reason beyond format compliance and instruction-following. In response, we propose Path-Aware Reinforcement Learning (PARL), a reinforcement learning framework that enhances tool-use by incorporating trajectory-based reward assignment and historical interaction memory. Empirical results show that PARL significantly improves performance across all tasks, particularly in tool function discovery and counterfactual rule adaptation. These findings highlight the importance of path-sensitive optimization and context accumulation in developing more robust, generalizable LLM agents.

Limitations

Despite the significant performance gains achieved, our work has two main limitations. First, in line with previous studies, we find that convergence during the reinforcement-learning phase remains highly sensitive to reward shaping, and the training dynamics can oscillate without careful tuning. We release all hyper-parameters to facilitate reproducibility and further adjustment, but model performance still hinges on precise reward scaling. Second, our experiments are constrained by limited computational resources to models of 7 B parameters or fewer. The response of larger models to RL signals—and their ability to generalise in more complex settings—has yet to be systematically evaluated. This will be a key focus of future work.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, and 1 others. 2024. Nestful: A benchmark for evaluating llms on nested sequences of api calls. *arXiv preprint arXiv:2409.03797*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and 1 others. 2024. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45.

Jie Chen, Xintian Han, Yu Ma, Xun Zhou, and Liang Xiang. 2024. Unlock the correlation between supervised fine-tuning and reinforcement learning in training code large language models. *arXiv preprint arXiv:2406.10305*.

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. *arXiv preprint arXiv:2504.11536*.

Hiroki Furuta, Yutaka Matsuo, Aleksandra Faust, and Izzeddin Gur. 2023. Exposing limitations of language model agents in sequential-task compositions on the web. *arXiv preprint arXiv:2311.18751*.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18030–18038.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Alex Gu, Naman Jain, Wen-Ding Li, Manish Shetty, Yijia Shao, Ziyang Li, Diyi Yang, Kevin Ellis, Koushik Sen, and Armando Solar-Lezama. 2025. Challenges and paths towards ai for software engineering. *arXiv preprint arXiv:2503.22625*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models (2023). *arXiv preprint arXiv:2308.00675*.

Li Hu, Guoqiang Chen, Xiuwei Shang, Shaoyin Cheng, Benlong Wu, Gangyang Li, Xu Zhu, Weiming Zhang, and Nenghai Yu. 2025. Compileagent: Automated real-world repo-level compilation with tool-integrated llm-based agent system. *arXiv preprint arXiv:2505.04254*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and 1 others. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Minki Kang, Jongwon Jeong, and Jaewoong Cho. 2025. T1: Tool-integrated self-verification for test-time compute scaling in small language models. *arXiv preprint arXiv:2504.04718*.

Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Fahad Shahbaz Khan, and Salman Khan. 2025. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36:43447–43478.

Ne Luo, Aryo Pradipta Gema, Xuanli He, Emile van Krieken, Pietro Lesci, and Pasquale Minervini. 2025. Self-training large language models for tool-use without demonstrations. *arXiv preprint arXiv:2502.05867*.

Bohan Lyu, Yadi Cao, Duncan Watson-Parris, Leon Bergen, Taylor Berg-Kirkpatrick, and Rose Yu. 2024. Adapting while learning: Grounding llms for scientific problems with intelligent tool usage adaptation. *arXiv preprint arXiv:2411.00412*.

Xinyi Ni, Qiuyang Wang, Yukun Zhang, and Pengyu Hong. 2025. Toolfactory: Automating tool generation by leveraging llm to understand rest api documentations. *arXiv preprint arXiv:2501.16945*.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Philip Resnik. 2025. Large language models are biased because they are large language models. *Computational Linguistics*, pages 1–21.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Z Shi, S Gao, X Chen, Y Feng, L Yan, H Shi, D Yin, Z Chen, S Verberne, and Z Ren. Chain of tools: Large language model is an automatic multi-tool learner (2024). *arXiv preprint arXiv:2405.16533*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv.org/abs/2303.11366*.

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

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025. Otc: Optimal tool calls via reinforcement learning. *arXiv preprint arXiv:2504.14870*.

Qiancheng Xu, Yongqi Li, Heming Xia, Fan Liu, Min Yang, and Wenjie Li. 2025. Petoolllm: Towards personalized tool learning in large language models. *arXiv preprint arXiv:2502.18980*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Seungbin Yang, ChaeHun Park, Taehee Kim, and Jaegul Choo. 2024b. Can tool-augmented large language models be aware of incomplete conditions? *arXiv preprint arXiv:2406.12307*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models, 2023. *URL https://arxiv.org/abs/2210.03629*.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2024. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *arXiv preprint arXiv:2410.07745*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

A Dataset Construction

As described in Section 4, we construct two tool-oriented datasets. The quantity and data splits of each dataset are shown in Table 5. The detailed construction process of the datasets is outlined below.

A.1 Rule Understanding Dataset

We first define three rule regimes. The Natural regime follows conventional operator precedence; the Linear regime processes every operator strictly left-to-right; the Reverse regime applies addition and subtraction before multiplication and division once parentheses are resolved. We synthesise expressions with a probabilistic context-free grammar. Operator probabilities are set to 0.30 for addition, 0.25 for subtraction, 0.30 for multiplication, and 0.15 for division. Expression complexity rises from two to more than six operators, and we raise the chance of inserting brackets from 0.20 to 0.60 as complexity grows. Operands are sampled uniformly from the interval $[-20, 20]$ for basic expressions and $[-50, 50]$ for intermediate and advanced ones. Nested brackets never exceed depth three, and a deterministic executor produces the gold answers under each rule. In total we generate 5 000 training and 1 000 test expressions, applying the above three rules to each expression.

A.2 Tool-Combination Dataset

We design nine synthetic APIs grouped into four functional themes: person information, date queries, price queries, and relation analysis. Each API accepts JSON input and yields a short textual observation. To create a task, we sample one to four relevant APIs, instantiate their parameters from a curated entity pool, then write a natural-language question whose answer demands concatenating the returned observations. All questions are verified automatically to ensure the tool sequence executes without error and that the final answer is unique and deterministic. The corpus contains 2,400 training and 600 test samples. Training data include 800 single-tool, 800 dual-tool, and 800 multi-tool queries; the test set mirrors this 1-1-1 balance with 200 instances per tier. We release generation code and corpora upon publication to enable full reproducibility. Get_financial_statement tool accepts a person's name as input and returns a description of that person's financial situation. Query_item_price tool returns the unit price of the product according to the product name, which is used as the basis for subsequent calculations. Query_data tool queries the month corresponding to the subject context, which can be used as a condition to filter the information obtained from the get_financial_statement tool. Get_relationship_chain tool receives a person's name and returns the corresponding relationship chain, which is used for dual-tool and multi-tool queries construction. Definition details of the four retrieval tools as shown in figure 6, The model needs to plan the tool usage order and query relevant information before arriving at the correct answer.

B Hyperparameters Setting

During the reinforcement learning training process, we used eight NVIDIA A100 GPUs together with the vLLM (Kwon et al., 2023) framework to accelerate both training and inference. We set the number of epochs to three and employed early stopping to prevent reward hacking. The learning rate was 1 e-6, gradient accumulation steps were 2, and the number of generations was 8. We used a batch size of 16 for both the Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct models, and likewise a batch size of 16 for the Qwen2.5-1.5B-Instruct and Qwen3-1.7B models. All experiments utilized DeepSpeed's ZeRO-3 (Rajbhandari et al., 2020)optimization.

For specific reward settings, the format reward is usually set to 1. If the model has a strong ability to follow instructions, the reward value should be appropriately lowered, otherwise it will affect the model's ability to summarize and reflect. The correctness reward is divided into two levels. When the model's answer is completely matched, the reward value is set to 3, because the result will be correct only when all steps are correct, so a larger reward value is given. When the model solves the problem correctly but does not answer according to the specified requirements, the reward value is set to 0.2. When the model correctly calls the tool in the required question, the reward value is set to 1. If the model incorrectly calls the tool, the reward value is set to 0.005, which is used to encourage the model to summarize the reasons for the error and try again. When the information of the tool is reduced, the model tends to refuse to answer. At this time, the two reward values for tool calls need to be adjusted up to train normally.

Split	Rule Understanding Dataset			Tool-Combination Dataset		
	Nature	Linear	Reverse	Single-Tool	Dual-Tool	Multi-Tool
Train	5k	5k	5k	800	800	800
Test	1k	1k	1k	200	200	200
All	6k	6k	6k	1k	1k	1k

Table 5: The quantity and data splits of two datasets.

C System Prompt

In system prompt, we used the <reasoning> and <answer> tags to encourage the model to think. In order to enable the model to fully learn how to use the tools, we then introduced in detail the purpose and specific usage rules of each tool. Finally, we explained the complete question-and-answer format, including the detailed format of calling tools and answer specifications. The complete system prompt is shown in Table 6.

D Case Study

When the model calls a simple tool, it can succeed once. However, when the input requirements of the tool are more complex, the model needs to modify the input based on the feedback of the tool call failure and then call the tool again. In other words, the model can correct itself through tool feedback and learn the correct tool call method. An example is shown in figure 7.

In the early stages of training, the model cannot handle dual-tool and multi-tool queries correctly. As shown in figure 8, the model tries to use one tool to obtain information that can only be obtained by calling two tools in sequence. However, due to the setting of reward functions for different tools, the model can learn how to split complex problems and call tools sequentially during training. An example of a model correctly splitting a problem in the later stages of training is shown in figure 12.

After removing the detailed description of the tool, the model showed a strong tendency to abandon the use of the tool. Even though the tool name already suggested that the tool could be used to obtain the corresponding information, the model still refused to use it. An example is shown in figure 9.

After removing the detailed description of the tool, the tool becomes more difficult to understand. After the model gets the result of the tool call, it needs to further understand the purpose of the tool based on the result. Therefore, in the early stage of training, it is difficult to take into account both the ability to understand the tool and the ability to summarize and analyze the tool. The tool is called correctly but errors are made in the summary. As shown in the figure 10, it is incredible that the model actually got an outrageous result of 2.8*5=15.2!

During the training process, the model can learn to call tools and use them through tool call rewards. On this basis, the model can improve its summary and planning capabilities through correct result rewards. Therefore, after removing the description, the model starts to try different tools through training, and plans the next path based on the tool feedback. After obtaining all the information, it can summarize and answer the question correctly. An example is shown in figure 11.

```
{
  "name_for_human": "Get Financial Statement",
  "name_for_model": "get_financial_statement",
  "description_for_model": "Get a person's pocket money.
  Format parameters as a JSON object.",
  "parameters": [
    {
      "name": "Personal",
      "description": "The person to whom the pocket money belong.",
      "required": True,  # Set to False if it is an optional parameter.
      "schema": {"type": "str"},
    }
  ]
}
```

```
{
  "name_for_human": "Query Date",
  "name_for_model": "query_date",
  "description_for_model": "Query the month in which the person is
  currently.  Format parameters as a JSON object.",
  "parameters": [
    {
      "name": "Personal",
      "description": "The name of the person.",
      "required": True,  # Set to False if it is an optional parameter.
      "schema": {"type": "str"},
    }
  ]
}
```

```
{
  "name_for_human": "Query Item Price",
  "name_for_model": "query_item_price",
  "description_for_model": "Query the unit price of an item.
  Format parameters as a JSON object.",
  "parameters": [
    {
      "name": "Item",
      "description": "The name of the item. Please retain
    the complete entity expression after the preposition of",
      "required": True,  # Set to False if it is an optional parameter.
      "schema": {"type": "str"},
    }
  ]
}
```

```
{
  "name_for_human": "Get Relationship Chain",
  "name_for_model": "get_relationship_chain",
  "description_for_model": "Get someone's relationship chain.
  Format parameters as a JSON object.",
  "parameters": [
    {
      "name": "Personal",
      "description": "The name of the person.",
      "required": True,  # Set to False if it is an optional parameter.
      "schema": {"type": "str"},
    }
  ]
}
```

Figure 6: Tool card of Tool-Combination Dataset.

 # Question:
Michael bought three rolls of stickers. How much did he spend in total? Please think step by step.

<reasoning>
Thought and Plan: The question provides the information that Michael bought three rolls of stickers, but it does not specify the price of a single roll. Therefore, we need to determine the price of a roll to calculate the total cost.
Action: query_item_price
Action Input: {"Item": "roll of stickers"}
Observation: Query failed. Please retain the complete entity expression after the preposition 'of'.
Thought and Plan: The query for the price of a roll of stickers failed. I need to try a different approach to find the price of a roll of stickers.
Action: query_item_price
Action Input: {"Item": "stickers"}
Observation: For stickers, The unit price is $3.7.
Final Answer: Michael spent $3.7 * 3 = $11.1 on three rolls of stickers.
</reasoning>
<answer>
Michael spent $11.1 on three rolls of stickers.
</answer>

Figure 7: The model correct its tool use methods in generating path.

Instruction Format

 # Instruction:
A conversation between User and Assistant. The user asks a question, and the Assistant solves it.The assistant first thinks about the reasoning process in the mind and then provides the user with the answer.The reasoning process and answer are enclosed within <reasoning> </reasoning> and <answer></answer>tags, respectively, i.e.,

```
<reasoning>
reasoning process here
</reasoning>
<answer>
answer here
</answer>
```

 # Available Tools:
To answer the questions correctly, you can choose to call the tools. You need to try to understand the function of each tool.Proper use of all tools can achieve functions such as calculation and information acquisition, and each tool has a specific function.Please try to explore the functions of different tools and the correct parameter input. You have access to the following APIs:

```
{name_for_model}: Call this tool to interact with the {name_for_human} API. What is the {name_for_human} API useful for? {description_for_model} Parameters: {parameters}
```

Use the following format:

 # Format Specifications:
Question: the question asked by the user
<reasoning>
 # Tool Usage Guidelines:

```
Thought: you should always think about what to do Action: the action to take, should be one of [{name_for_model}, ......] Action Input: the input to the action Observation: the results of the actions
```

Note: If you need to call the tool again (re-call the same tool or call a new tool), repeat the following structure: (this Thought/Action/Action Input/Observation),that is, this structure: Thought/Action/Action Input/Observation can be repeated zero or more times.
</reasoning>
<answer>
Thought: I now know the final answer
Final Answer: the final answer to the original input question
</answer>

Begin!

Table 6: Structured reasoning and tool-use interaction format for Assistant responses.

Question:
How much pocket money did Gregory's classmate have in June? Please think step by step.

<reasoning>
Thought and Plan: The question asks about the amount of pocket money Gregory's classmate had in June. However, there is no specific information provided about Gregory's classmate or the amount of pocket money. Therefore, I need to determine if there is any available data or if I need to use a tool to find the answer.

Thought and Plan: The query for 'Pocket money' failed. I need to try another approach to find the answer.
Action: query_date
Action Input: {"Personal": "Gregory's classmate"}
Observation: Query failed.

Thought and Plan: The query for 'Gregory's classmate' failed. I need to try another approach to find the answer.
Action: get_relationship_chain
Action Input: {"Personal": "Gregory's classmate"}
Observation: Query failed.

Thought and Plan: The query for 'Gregory's classmate' failed. I need to try another approach to find the answer.
Action: get_financial_statement
Action Input: {"Personal": "Gregory's classmate"}
Observation: Query failed.

Thought and Plan: The query for 'Pocket money' failed. I need to try another approach to find the answer.
Action: query_date
Action Input: {"Personal": "Gregory's classmate"}
Observation: Query failed.

Figure 8: The model tries the tools and aggregates the feedback to get the correct answer.

Question:
Robert bought one dozen erasers. How much did he spend in total? Please think step by step.

<reasoning>
Thought and Plan: The question is asking about how much Robert spent in total after buying one dozen erasers. However, the provided tools do not include any that can calculate the cost of items based on their price per unit. The available tools are for financial statements, item prices, dates, and relationship chains. Since no tool is available to calculate the total cost, the question cannot be answered using the provided tools.
Action: None
Action Input: None

Figure 9: After removing the description, the initial training model refuses to use the tool.

Question:
Michael bought five packs of file folders. How much did he spend in total? Please think step by step.

<reasoning>
Thought and Plan: The question asks for the total amount Michael spent when he bought five packs of file folders. However, the question does not provide the price of a single pack of file folders. Therefore, we need to use a tool to get the price of a pack of file folders.
Action: query_item_price
Action Input: {"Item": "file folders"}
Observation: For file folders, The unit price is $2.8.

Thought and Plan: Now that we know the unit price of file folders, we can calculate the total cost by multiplying the unit price by the number of packs Michael bought.
Action: get_financial_statement
Action Input: {"Personal": "Michael"}
Observation: Michael's pocket money in January was $716. Michael's pocket money in February was $516. ······
Thought: The information about Michael's pocket money in the financial statement does not help in calculating the total amount he spent on file folders. The tool is not applicable for this question.

Final Answer: Michael spent $15.2 on file folders.
</reasoning>
<answer>
Michael spent $15.2 on file folders.
</answer>

Figure 10: The model is trained to correctly call the tool without the description, but an error occurs when summarizing.

 # Question:
Michael bought three bundles of rulers. How much did he spend in total? Please think step by step.

<reasoning>
Thought: The question provides information that Michael bought three bundles of rulers, but it does not specify the price per bundle or the total cost. Therefore, there is not enough information to calculate the total cost.
Action: tool_A
Action Input: {"Param_A": "Michael"}
Observation: Michael's pocket money in January was $716. Michael's pocket money in February was $516······

Thought: The provided information about Michael's pocket money does not relate to the question about the cost of three bundles of rulers.
Action: tool_B
Action Input: {" Param_A ": "Michael"}
Observation: Query failed. Please retain the complete entity expression after the preposition 'of'.

Thought: The question is about the total cost of three bundles of rulers, but no information is provided about the price per bundle. Therefore, it is impossible to calculate the total cost.
Action: tool_C
Action Input: {" Param_A ": "Michael"}
Observation: Michael spent all his pocket money on a toy in January.

Thought: The provided information about Michael's pocket money does not relate to the question about the cost of three bundles of rulers.
Action: tool_D
Action Input: {" Param_A ": "Michael"}
Observation: Michael's classmate is Robert.

Thought: The provided information about Michael's classmate does not relate to the question about the cost of three bundles of rulers.
Action: tool_B
Action Input: {"Param_A": "rulers"}
Observation: For rulers, The unit price is $6.5.

Thought: The total cost of three bundles of rulers is 3 * $6.5 = $19.5.
</reasoning>
<answer>
Final Answer: $19.5
</answer>

Figure 11: The problem of using two tools cannot be solved in the early stage of model training

Question:
How much pocket money did Edward's neighbor have in August? Please think step by step.

<reasoning>
Thought and Plan: The question asks for the amount of pocket money Edward's neighbor had in August. However, there is no direct information provided about the neighbor's pocket money or any specific data related to August. Therefore, I need to determine if there is any available information or if I need to use a tool to find the answer.
Action: tool_A
Action Input: {"Param_A": "Edward"}
Observation: Edward's pocket money in January was $867. Edward's pocket money in February was $661······

Thought and Plan: The information provided in the observation shows Edward's pocket money in August. However, the question asks about Edward's neighbor's pocket money in August. Therefore, I need to determine if there is any additional information or if I need to use another tool to find the answer.
Action: tool_B
Action Input: {"Param_A": "Edward"}
Observation: Query failed.

Thought and Plan: The information provided in the observation shows Edward's pocket money in August. However, the question asks about Edward's neighbor's pocket money in August. Therefore, I need to determine if there is any additional information or if I need to use another tool to find the answer.
Action: tool_C
Action Input: {"Param_A": "Edward"}
Observation: Edward spent all his pocket money on a toy in September

Thought and Plan: The information provided in the observation shows Edward's pocket money in August. However, the question asks about Edward's neighbor's pocket money in August. Therefore, I need to determine if there is any additional information or if I need to use another tool to find the answer.
Action: tool_D
Action Input: {"Param_A": "Edward"}
Observation: Edward's neighbor is Ronald.

Thought and Plan: The information provided in the observation shows Edward's pocket money in August. However, the question asks about Edward's neighbor's pocket money in August. Therefore, I need to determine if there is any additional information or if I need to use another tool to find the answer..
Action: : tool_A
Action Input: {"Param_A": " Ronald"}
Observation: Ronald's pocket money in January was $702. Ronald's pocket money in February was $829······

Thought and Plan: The information provided in the observation shows Ronald's pocket money in August. Therefore, the answer to the question is $752.
Final Answer: $752
</reasoning>
<answer>
Final Answer: $752
</answer>

Figure 12: The model tries the tools and collects feedback to correctly solve the problem of removing descriptions and combining tools