Teaching LLMs to Learn Tool Trialing and Execution through Environment Interaction

Anonymous ACL submission

Abstract

Equipping Large Language Models (LLMs) with external tools enables them to solve complex real-world problems. However, the robustness of existing methods remains a critical challenge when confronting novel or evolving tools. Existing trajectory-centric paradigms primarily rely on memorizing static solution paths during training, which limits the ability of LLMs to generalize tool usage to newly introduced or previously unseen tools. In this paper, we propose ToolMaster, a framework that shifts tool use from imitating golden tool-calling trajectories to actively learning tool usage through interaction with the environment. To optimize LLMs for tool planning and invocation, ToolMaster adopts a trial-and-execution paradigm, which trains LLMs to first imitate teacher-generated trajectories containing explicit tool trials and self-correction, followed by reinforcement learning to coordinate the trial and execution phases jointly. This process enables agents to autonomously explore correct tool usage by actively interacting with environments and forming experiential knowledge that benefits tool execution. Experimental results demonstrate that ToolMaster significantly outperforms existing baselines in terms of generalization and robustness across unseen or unfamiliar tools. All code and data will be released via GitHub.

1 Introduction

Large Language Models (LLMs) have demonstrated strong planning and reasoning capabilities, and equipping them with external tools has proven effective in further enhancing their ability to solve complex, real-world problems (Qin et al., 2024; Song et al., 2023a; Qin et al., 2023). Typical examples include using calculators to improve numerical accuracy (Cobbe et al., 2021; Parisi et al., 2022) and leveraging search engines to retrieve factual knowledge (Carlini et al., 2021; Thoppilan et al., 2022; Borgeaud et al., 2022). To further enhance the tool-use capability of LLMs, recent studies have primarily focused on enabling LLMs to perform more effective tool planning, tool decision, and reliable tool invocation (Shen et al., 2023; Lu et al., 2023; Liang et al., 2023).

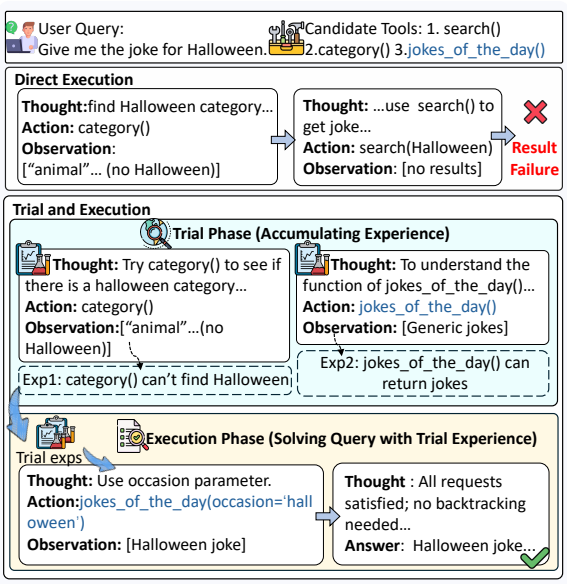


Figure 1: Illustration of the Trial-and-Execution paradigm proposed by ToolMaster.

Earlier works (Yao et al., 2023; Song et al., 2023a) mainly rely on prompting-based methods to decompose tool-learning tasks into sub-tasks and generate grounded plans by leveraging the reasoning capabilities of LLMs (Wei et al., 2022). However, such approaches remain significantly limited in their tool-use performance (Qin et al., 2024). To overcome these limitations, recent research on tool-using agents has increasingly adopted a trajectory-centric post-training paradigm (Qin et al., 2024; Tang et al., 2023; Qian et al., 2025; Feng et al., 2025; Yu et al., 2025). The dominant approach typically involves collecting high-quality trajectories through sampling methods (e.g., MCTS) or

expert demonstrations, followed by Supervised Fine-Tuning (SFT) (Qin et al., 2024) or Reinforcement Learning (RL) (Qian et al., 2025) to enforce imitation of desired trajectories. While effective on fixed toolsets, this trajectory-centric paradigm brute-forces LLMs to imitate specific tool-use trajectories, causing them to struggle when tools evolve or when deployment scenarios deviate from these supervisions (Zeng et al., 2025).

To enhance the tool-use generalization capability of LLMs, existing methods typically leverage their self-reflection and self-asking abilities to enable more accurate tool planning and invocation (Mekala et al., 2024; Ma et al., 2025). However, a fundamental challenge remains: LLMs often lack robustness in real-world applications when explicit feedback from environments is unavailable (Wang et al., 2024). As illustrated in Figure 1, when an LLM is presented with a newly introduced tool "jokes_of_the_day()", which is required to solve the task, alongside a well-learned tool "search()" that fails to return relevant knowledge, the model may still directly invoke "search()", leading to incorrect results. This behavior likely arises because "search()" appears frequently in the training data, making the LLM overly confident in invoking it. In contrast, by benefiting from tool trials that enable tool-usage experiments through interactions with the environment, the LLM is able to perform correct tool invocation and obtain accurate results, highlighting the necessity of tool trialing in tool learning tasks.

In this paper, we build ToolMaster upon the Trial-and-Execution paradigm, aiming to fully exploit tool-calling feedback from the environment prior to tool planning and invocation. Specifically, ToolMaster first optimizes LLMs via supervised fine-tuning to imitate tool-trialing behaviors using tool-calling trajectories generated by a teacher model. Subsequently, we employ reinforcement learning to further optimize the model, enabling it to jointly coordinate tool trialing and tool execution actions for more accurate outcomes. During tool-calling trajectory synthesis, we adopt a more capable LLM as the teacher model and prompt it to perform tool trialing by invoking tools to obtain feedback, followed by the tool execution phase that leverages the accumulated experiences for explicit tool planning and self-correction.

Our experiments on three different tool-learning datasets demonstrate the effectiveness of ToolMaster, which achieves more than a 7% improvement over baseline models. Benefiting from the trial phase, LLMs perform more tool-calling interaction steps, which significantly reduces execution failures and leads to higher accuracy. Moreover, ToolMaster exhibits strong generalization capability, reflected in its accuracy in both unfamiliar tool-calling scenarios and problem-solving that requires previously unseen tools. Notably, ToolMaster also alleviates unnecessary biases when invoking tools that are rarely observed in the training dataset.

2 Related Work

Tool use extends the capabilities of Large Language Models (LLMs) by allowing them to interact with external environments. Early paradigms for tool use, such as RestGPT (Song et al., 2023a) and ReAct (Yao et al., 2023), relied on in-context learning to prompt LLMs to leverage tools for problem solving. To further enhance tool use capabilities, ToolLLM (Qin et al., 2024) employs supervised fine-tuning of LLMs using the collected dataset ToolBench, which contains large-scale tool usage trajectories constructed via the Depth-First Search-based Decision Trees (DFSDT) method. However, such SFT-based methods that rely on curated trajectories tend to overfit the training signals and suffer from catastrophic forgetting (Luo et al., 2023).

Instead of SFT, recent works have further employed reinforcement learning methods to optimize LLMs to enhance their capabilities in tackling complex tool-use tasks. TP-LLaMA (Chen et al., 2024) applies Direct Preference Optimization (DPO) (Rafailov et al., 2023) to align models with preferred tool paths, as well as ToolRL (Qian et al., 2025) and Tool-Zero (Zeng et al., 2025) further leverage the Group Relative Policy Optimization (GRPO) method (Shao et al., 2024) to optimize LLMs, emphasizing the design of sophisticated reward functions for guidance, such as the accuracy of tool calls along ground-truth trajectories. Furthermore, to ensure the effectiveness of training, FTRL (Ye et al., 2025b) proposes a stable and verifiable method for synthesizing tool-use training data. However, these methods may suffer from the reward hacking problem (Skalse et al., 2022), where the model performs fewer trials to maximize the tool-calling accuracy reward, thereby limiting their generalization across different tool-use scenarios (Mekala et al., 2024).

To enhance the generalization ability of LLMs in tool use, substantial efforts have been directed

towards enhancing their capability to use new tools and incorporating the tool execution feedback for self-correction. Some works (Mekala et al., 2024) employ self-asking contrastive questions for tool selection and parameter generation to fully exploit new tools. Other methods (Ma et al., 2025) introduce self-correction mechanisms by reflecting on errors in the tool-calling trajectories based on feedback from the tool executions. However, these approaches mainly focus on correcting tool-calling errors using environmental feedback, neglecting the proactive agentic role of LLMs in autonomously conducting tool-calling trials for planning and reasoning.

3 Methodology

In this section, we present ToolMaster, a framework illustrated in Figure 2 that optimizes LLMs to master tools through trial-and-execution paradigm. Specifically, the model first conducts pre-interactions in the trial phase to accumulate tool-usage experience and refine its internal belief. After this initial calibration, the model proceeds to the execution phase, where it leverages the tool-usage experience to iteratively solve the task through self-correction. To realize this paradigm, we first introduce the optimization pipeline (Sec. 3.1), detailing the training strategy used to instill such reasoning capabilities. To facilitate this training, we then detail the underlying data synthesis methodology (Sec. 3.2), explaining how we prompt LLMs to generate the requisite high-quality tool trial trajectories.

3.1 Optimizing LLM Tool Exploration Capability via Environment Interaction

To enhance the tool planning capability, we optimize LLMs to perform more effective tool exploration by conducting trials to execute tools within the environment $\mathcal{E}$.

Tool Planning with Environment Feedback. Given a user query q and a set of candidate tools $\mathcal{S}_{\text{Tool}}$, the tool learning task requires the LLM to perform tool planning, invoke appropriate APIs, and interact with the external environment to obtain an intermediate reasoning result to generate the final answer y:

$$\tau, y = \text{LLM}(q, \mathcal{S}_{\text{Tool}}), \tag{1}$$

where the reasoning trajectories τ can be represented:

$$\tau = \{(r_1, a_1, o_1), (r_2, a_2, o_2), \ldots, (r_N, a_N, o_N)\}, \tag{2}$$

where r_i denotes the reasoning step at time i, $a_i \in \mathcal{S}_{\text{Tool}}$ represents the action to trigger an API call with structured arguments, and o_i is the observation returned by the environment after executing a_i.

At each step i, the LLM conditions on the accumulated context to produce reasoning and actions:

$$p(r_i, a_i \mid q, h_{i-1}), \tag{3}$$

where the history $h_{i-1} = \{(r_j, a_j, o_j)\}_{j=1}^{i-1}$ includes all previous reasoning steps, tool calls, and observations. The environment executes the selected tool action and returns an observation:

$$o_i = \mathcal{E}(a_i), \tag{4}$$

which is appended to the context for subsequent reasoning. This iterative reasoning-action-observation loop enables the model to decompose complex queries into a sequence of grounded tool invocations, allowing explicit interaction with the environment $\mathcal{E}$. However, such a tool planning paradigm does not fully exploit the feedback signals returned by the environment during tool invocation, as it fails to explicitly incorporate essential environment feedback into the tool planning process.

Trajectory Optimization Strategies. To enhance the tool exploration capability of LLMs, we first optimize the model to acquire richer tool exploration behaviors by distilling reasoning trajectories from a superior LLM. Subsequently, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is employed to maximize execution success within the environment $\mathcal{E}$.

First, ToolMaster prompts a superior LLM to generate tool trial trajectories by explicitly instructing tool trials and self-correction behaviors, thereby constructing the SFT dataset $\mathcal{D}$ (Sec. 3.2). For each trajectory $\tau \in \mathcal{D}$, we optimize the LLM parameters θ by minimizing the following loss:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_{i=1}^{N} \log \pi_\theta(r_i, a_i \mid q, h_{i-1}) \right]. \tag{5}$$

We then adopt a composite outcome-based reward function to maximize the expected cumulative reward during LLM optimization:

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E}_{\tau \sim \pi_\theta}[R_{\text{fmt}} + R_{\text{corr}}], \tag{6}$$

where $R_{\text{fmt}} \in \{0, 1\}$ denotes the format reward, which is set to 1 if the model strictly follows the prescribed reasoning and tool invocation schema, and $R_{\text{corr}} \in \{0, 1\}$ denotes the answer correctness reward, which is assigned a value of 1 if the final

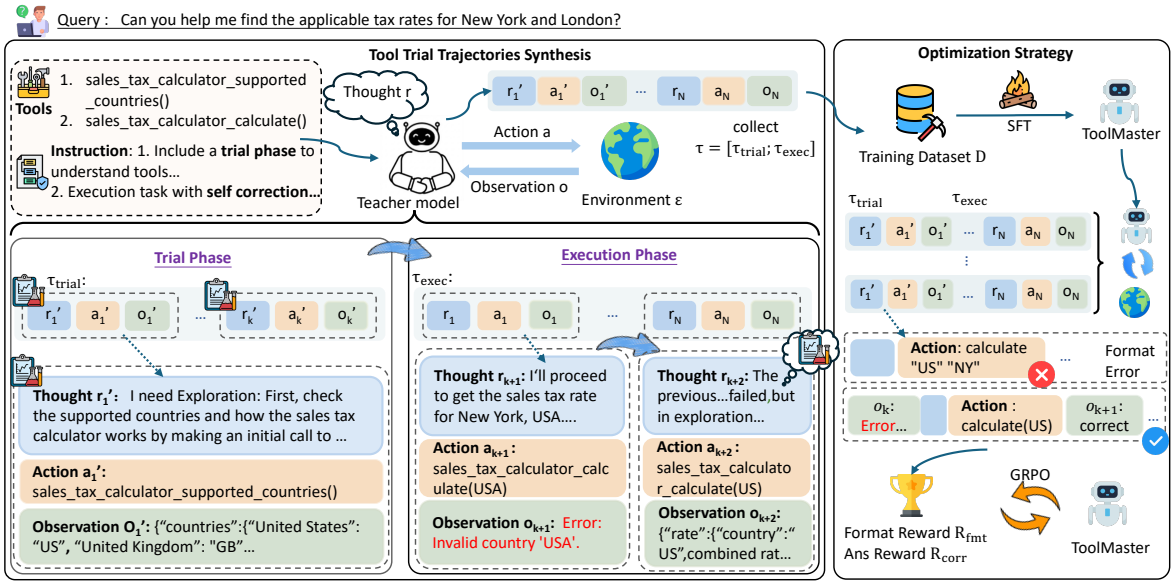


Figure 2: The architecture of ToolMaster.

answer correctly resolves the user query. To estimate the answer correctness, we employ a stronger LLM as an automatic judge to evaluate the tool execution results and compute the reward R_{corr}.

3.2 Prompting LLMs to Synthesize Tool Trial Trajectories for SFT

In this subsection, we describe the methodology for constructing the SFT dataset $\mathcal{D}$ by synthesizing high-quality training trajectories that autonomously encourage LLMs to explore tool usage under the *trial-and-execution* paradigm. Specifically, we employ a strong reasoning-capable model, such as DeepSeek-V3.1 (DeepSeek-AI, 2024), as the teacher π_{teacher}, and prompt it to fully engage in tool-usage trials while collecting rich feedback from tool invocations. The feedback from the environment is incorporated as a part of the tool-use experience, which is leveraged to enhance both tool planning and task-solving capabilities.

Tool Trialing with Environment. In the trial phase, the model fully interacts with the environment to collect sufficient feedback on tool usage:

$$\tau_{\text{trial}} \sim \pi_{\text{teacher}}(\cdot \mid \mathcal{I}, q, \mathcal{S}_{\text{Tool}}), \tag{7}$$

where $\mathcal{I}$ is the instruction. The trajectory τ_{trial} contains k autonomous tool-calling trials, which are determined by π_{teacher}:

$$\tau_{\text{trial}} = \{(r'_j, a'_j, o'_j)\}_{j=1}^{k}, \tag{8}$$

where each tuple (r'_j, a'_j, o'_j) corresponds to an investigative step rather than a direct solution attempt. Specifically, the reasoning thought r'_j formulates a hypothesis to explore particular tool semantics or parameter constraints; the tool action a'_j executes a probing operation to verify functional behaviors; and the observation o'_j reveals the environmental feedback during tool invocation. Through these interactions, the model accumulates grounded observations with the trajectory τ_{trial} as empirical experience, allowing it to calibrate its understanding of the available toolset within the actual environment prior to the tool planning and execution phase.

Tool Execution. Given the tool-use experience τ_{trial}, the model generates an execution trajectory τ_{exec} for tool planning and invocation:

$$\tau_{\text{exec}} \sim \pi_{\text{teacher}}(\cdot \mid \mathcal{I}, q, \mathcal{S}_{\text{Tool}}, \tau_{\text{trial}}), \tag{9}$$

where the problem-solving trajectory τ_{exec} contains N tool invocations to resolve the query q:

$$\tau_{\text{exec}} = (r_j, a_j, o_j)_{j=1}^{N}, \tag{10}$$

where each (r_j, a_j, o_j) denotes an execution step. Specifically, r_j analyzes the current context to formulate a solution strategy, a_j executes a purposeful operation to advance the task, and o_j represents the feedback from the environment. To enable the teacher model to effectively leverage error signals from the environment, we incorporate a self-correction mechanism that rectifies the intermediate state, ensuring that the trajectory is guided back toward the correct final answer. Consequently, the final answer y is given by:

$$y \sim \pi_{\text{teacher}}(\cdot \mid \mathcal{I}, q, \mathcal{S}_{\text{Tool}}, \tau_{\text{trial}}, \tau_{\text{exec}}). \tag{11}$$

SFT Data Curation. Finally, we collect the SFT dataset $\mathcal{D}$ for SFT. Formally, for each query q, we construct a trial-and-execution trajectory τ by concatenating the sub-trajectories from the trial and execution phases:

$$\tau = [\tau_{\text{trial}}; \tau_{\text{exec}}]. \tag{12}$$

Subsequently, the final SFT dataset $\mathcal{D}$ is obtained by filtering for high-quality trajectories:

$$\mathcal{D} = \{(q_1, \tau_1, y_1), \ldots, (q_K, \tau_K, y_K)\}, \tag{13}$$

where only the trajectories τ that successfully resolve the corresponding query q and strictly adhere to the behavioral guidelines specified in $\mathcal{I}$ are retained. Further details of the filtering methodology are provided in Appendix A.2.

4 Experimental Methodology

This section describes datasets, baselines, and implementation details used in experiments.

Datasets. To construct our training data, we leverage the training split of the publicly available ToolBench (Qin et al., 2024). Specifically, we curate a subset of 1,500 queries to construct the SFT dataset and 800 queries for RL training. To ensure a comprehensive evaluation, we employ one in-domain benchmark: (1) StableToolbench (Guo et al., 2024), a stabilized suite covering diverse domains and multi-tool compositions; and two Out-of-Domain (OOD) benchmarks to assess generalization: (2) TMDB (Song et al., 2023b), which tests precise API mapping and argument filling, and (3) ToolHop (Ye et al., 2025a), which evaluates complex multi-hop reasoning and cross-tool planning.

Baselines. For a comprehensive evaluation, we benchmark ToolMaster against three distinct categories of baselines: (1) Zero-shot LLMs, (2) SFT-based baselines, and (3) RL-based methods.

First, for Zero-shot LLMs, we evaluate powerful proprietary and open-weights models, specifically GPT-4o (OpenAI et al., 2024), GPT-4o-mini (OpenAI, 2024), and Qwen2.5-32B-Instruct (Qwen, 2024). Additionally, we report the zero-shot performance of the backbone models employed in our training to serve as a direct baseline for quantifying improvement. Regarding SFT-based Baselines, we compare against Distill (SFT), which is fine-tuned on successful tool-use trajectories distilled from DeepSeek-V3.1 (DeepSeek-AI, 2024), and Tool-LLM (Qin et al., 2024), a robust data-centric approach that utilizes a Depth-First Search Decision Tree (DFSDT) to construct high-quality solution paths for instruction tuning. Finally, we compare against several RL-based methods, including StepTool (Yu et al., 2025), FTRL (Ye et al., 2025b), and ToolRL (Qian et al., 2025). Specifically, FTRL (Ye et al., 2025b) and ToolRL (Qian et al., 2025) are built upon the GRPO framework with different reward formulations, whereas StepTool (Yu et al., 2025) optimizes the policy with PPO and assigns step-wise rewards to enable explicit reward assignment for each intermediate tool-use step.

Evaluation Metrics. Following previous work Lu et al. (2025), for StableToolBench (Guo et al., 2024) and TMDB (Song et al., 2023b), we adopt the Solvable Pass Rate (SoPR) for evaluation. Following Ma et al. (2025), we leverage GPT-4o as the evaluator and utilize the same prompts to categorize responses into "Solved", "Unsolved", or "Unsure". A score of 1 is assigned to "Solved" instances, while others receive 0. For ToolHop (Ye et al., 2025a), we evaluate Answer Correctness based on whether the model's output contains the ground truth answer. More details of evaluation prompts and criteria are provided in Appendix A.3.

Implementation Details. We conduct experiments on three backbone models: Qwen2.5-7B-Instruct (Qwen, 2024), Qwen3-8B (Qwen, 2025), and Qwen3-14B (Qwen, 2025). For data synthesis, we employ DeepSeek-V3.1 (DeepSeek-AI, 2024) as the teacher to generate SFT trajectories, and subsequently utilize it as a verifier to perform data filtering for quality assurance. Regarding the training configuration, we first train the models in the SFT phase for 3 epochs with a learning rate of 1×10^{-5} and a maximum sequence length of 8,192. In the subsequent GRPO stage, we adopt a learning rate of 1×10^{-6}, set the KL coefficient to $\beta = 0.002$, and utilize a group size of 4 with the correctness reward R_{corr} determined by DeepSeek-V3 (DeepSeek-AI, 2024) based on its direct evaluation of the execution trajectorys task fulfillment. During inference, we set the temperature to 0.1 for all models to ensure consistent evaluation. Detailed implementations are reported in Appendix A.2.

5 Evaluation Results

In this section, we first present the overall performance of ToolMaster. Subsequently, we demonstrate its generalization capabilities in out-of-domain (OOD) settings, followed by ablation studies and analyses to validate the effectiveness of our

Backbone Model	Method	I1 Inst	I1 Cat	I1 Tool	I2 Cat	I2 Inst	I3 Inst	Avg.
GPT-4o	Zero-shot	59.28	56.21	64.56	61.29	60.38	60.66	60.23
GPT-4o-mini	Zero-shot	56.44	52.29	65.19	54.84	50.00	50.82	54.93
Qwen2.5-32B-Instruct	Zero-shot	54.60	52.29	56.96	55.65	48.11	50.82	53.07
LLaMA-2-7b-hf	ToolLLM	50.92	43.14	51.92	41.94	39.62	42.62	45.03
	StepTool	39.26	38.56	41.67	30.58	34.91	31.15	36.02
Qwen2.5-7B-Instruct	Zero-shot	49.08	44.44	50.63	41.94	36.79	37.70	43.43
	Distill (SFT)	55.83	56.21	56.96	54.84	50.00	49.18	53.84
	ToolLLM	52.15	54.25	55.70	50.81	47.17	57.38	52.91
	ToolRL	52.15	49.67	51.90	44.35	49.06	44.26	48.57
	FTRL	60.43	57.19	62.66	59.27	54.25	56.56	58.39
	ToolMaster	**66.26**	**65.36**	**65.82**	**66.13**	**64.15**	**70.49**	**66.37**
Qwen3-8B	Zero-shot	50.92	53.59	53.80	45.97	40.57	40.98	47.64
	Distill (SFT)	60.12	60.13	65.19	52.42	64.15	52.46	59.08
	ToolMaster	**63.19**	**64.71**	**66.46**	**66.94**	**67.92**	**68.85**	**66.34**
Qwen3-14B	Zero-shot	55.83	55.56	61.39	51.61	46.23	57.38	54.67
	Distill (SFT)	66.26	63.40	68.99	59.68	58.49	49.18	61.00
	ToolMaster	**69.33**	**67.32**	**72.78**	**70.97**	**75.47**	**70.49**	**71.06**

Table 1: Overall performance comparison of different methods on StableToolBench. The best results are highlighted in **bold**, and the second-best results are underlined.

Method	TMDB	ToolHop	Avg.
GPT-4o-mini	75.00	42.21	57.60
GPT-4o	80.00	45.32	61.81
Qwen2.5-32B-Instruct	73.00	20.00	46.50
Qwen2.5-7B-Instruct			
Zero-shot	69.00	16.68	42.84
Distill (SFT)	65.00	27.43	46.22
ToolLLM (2024)	68.00	33.96	50.98
FTRL (2025b)	56.00	29.05	42.53
ToolRL (2025)	67.00	32.46	49.73
ToolMaster	**86.00**	**37.38**	**61.69**
Qwen3-8B			
Zero-shot	79.00	38.59	58.80
Distill (SFT)	70.00	43.71	56.86
ToolMaster	**82.00**	**45.03**	**63.52**

Table 2: Performance on TMDB and ToolHop to evaluate the generalization capability of different methods.

proposed trial-and-execution paradigm.

5.1 Overall Performance

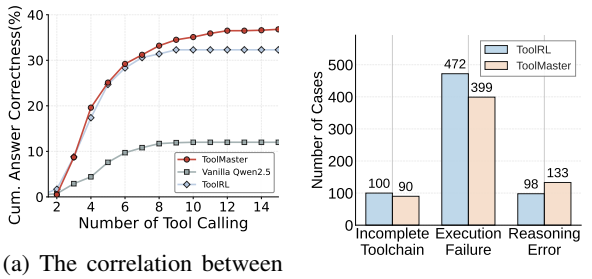
This subsection shows the overall performance of ToolMaster under both in-domain and out-of-domain testing settings.

We first present the main results of ToolMaster across all subsets of StableToolBench in Table 1. Overall, ToolMaster consistently outperforms all baseline methods, achieving an average improvement of more than 7% over baseline models. Notably, this advantage remains stable across different backbone models, demonstrating the generalization ability of ToolMaster. Compared with prompting-based methods, ToolMaster yields over 10% improvements, indicating that relying solely on prompting LLMs to enable tool use capability is less effective. In comparison with Distill (SFT), ToolMaster also achieves substantial gains, demonstrating the benefit of adopting more effective training strategies, such as RL methods, to better leverage supervision signals for guiding LLMs in tool usage. Furthermore, when compared with RL-based methods such as FTRL and ToolRL, ToolMaster shows improvements exceeding 7%, underscoring the effectiveness of the trial-and-execution paradigm, which constructs valuable tool-use experiences through iterative interaction with tools and then utilizes the experiences for tool planning and invocation.

To further validate the generalization capability of ToolMaster, we conduct evaluations under out-of-domain (OOD) settings, specifically assessing performance in tool-rich environments (TMDB) and complex multi-hop reasoning scenarios (ToolHop). As shown in Table 2, ToolMaster consistently outperforms all baselines, surpassing the strongest competitor by an average margin of 10%. This notable performance gain demonstrates the strong generalization ability of ToolMaster in unseen tool use scenarios. We attribute this advantage to the trial-based experiments, which enable the model to accumulate tool-use experience by actively testing tools in the environment. Such experience equips the model with more transferable tool-use knowledge, allowing it to effectively solve problems in previously unseen environments.

Method	I1	I2	I3	Avg.
ToolMaster	**65.81**	**65.15**	**70.49**	**66.37**
w/o SFT	60.41	57.06	55.74	58.52
w/o RL	59.70	60.90	57.38	59.72
w/o Trial-and-Exec	62.00	61.16	60.66	61.50
ToolMaster (SFT)	59.70	60.90	57.38	59.72
w/o Trial Phase	56.73	54.78	52.46	55.37
w/o Self-Correction	57.58	56.67	50.82	56.15
w/o Trajectory Filter	56.05	57.13	55.73	56.36

Table 3: Performance of components in ToolMaster. All models are implemented using Qwen2.5-7B-Instruct.



(a) The correlation between answer correctness and the number of tool calling.

(b) Distribution of tool-calling error types.

Figure 3: Tool-calling analyses of ToolMaster in out-of-domain scenarios. We use Qwen2.5-7B-Instruct as the backbone model in experiments and conduct experiments on the ToolHop dataset.

5.2 Ablation Study

In this subsection, we conduct ablation studies to evaluate the contribution of various components in ToolMaster and report the results across different difficulty levels (I1, I2, I3) of StableToolBench.

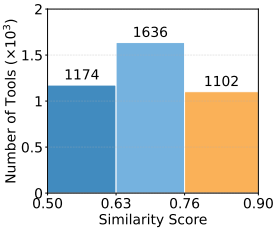
As shown in Table 3, we first examine the impact of different training strategies used by ToolMaster. Specifically, ToolMaster w/o SFT and ToolMaster w/o RL remove the SFT and RL processes, respectively, to assess their effectiveness. ToolMaster w/o Trial-and-Exec uses only the golden tool-use trajectories, omitting the trial-and-execution process during SFT training. Next, we investigate the role of the trial-and-execution mechanisms in constructing the SFT dataset. ToolMaster (SFT) w/o Trial Phase removes the tool trial process, while ToolMaster (SFT) w/o Self-Correction eliminates the self-correction step within the execution process. Additionally, ToolMaster (SFT) w/o Trajectory Filter is included to demonstrate the impact of the data filtering strategy during SFT dataset construction.

The evaluation results show that, when removing the SFT or RL training phase, the performance of ToolMaster degrades, demonstrating the necessity of both. Notably, the application of the Trial-and-Execution paradigm yields an additional improvement of approximately 5% (comparing ToolMaster with ToolMaster w/o Trial-and-Exec), as it enables the model to conduct tool-use trials, forming experiments that facilitate tool planning and invocation. Next, we analyze the roles of different components within the trial and execution phases in curating the SFT dataset. The results indicate that both the trial phase and self-correction within the execution phase effectively supervise LLMs for tool usage. The trial phase is particularly beneficial for easier tasks (I1 and I2), as it helps form effective experiments that improve tool-use accuracy, while self-correction proves more effective for more challenging tasks (I3), stimulating self-reflection to verify and refine the reasoning process. Furthermore, removing the data filter leads to performance degradation, validating the necessity of training on high-quality, filtered trajectories.

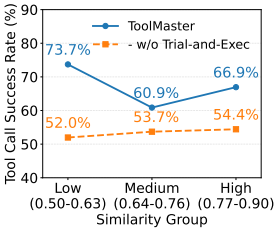
5.3 Effectiveness of ToolMaster in Out-of-Domain Tool-Calling Scenarios

As shown in Figure 3, we analyze the effectiveness of ToolMaster in out-of-domain tool-calling scenarios to evaluate its generalization capability and tool-calling behavior.

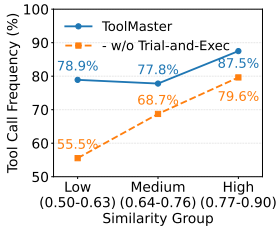
First, we examine tool trial effectiveness in Figure 3(a) by plotting accumulated answer correctness against the number of tool calling steps. In out-of-domain settings, the model is often unfamiliar with the tools required for the given problems; therefore, conducting appropriate tool calls is crucial for evaluating tool-learning methods. The results show that the correctness of all models increases sharply before the 7-th step, indicating that necessary tool-calling steps are essential for answering the questions. Tool calls typically occur during the execution stage and serve as critical intermediate steps for question answering. As the number of tool calls increases, both Vanilla LLM and ToolRL exhibit plateauing correctness, whereas ToolMaster continues to improve. This suggests that ToolMaster is able to conduct effective additional tool trials that better facilitate tool execution to produce accurate results. As shown in Figure 3(b), we further analyze the distribution of tool-calling errors across three categories: Incomplete Toolchain, Execution Failure, and Reasoning Error. These categories are judged using a stronger LLM, DeepSeek-V3.1. The results indicate that ToolMaster achieves the most notable improve-



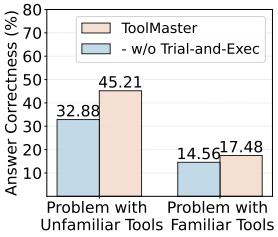
(a) Distribution of tools across different similarity levels.



(b) Tool calling success rates for different similarity groups.



(c) Tool calling frequency of test instances using tools with different similarity levels.



(d) Answer correctness for familiar and unfamiliar tools.

Figure 4: Characteristics of ToolMaster in tool usage under varying degrees of similarity to the training data. This experiment uses Qwen2.5-7B-Instruct as the backbone model and is evaluated on the ToolHop dataset. Figures 4(a) and 4(b) illustrate the distribution of tools and their calling success rates based on similarity between tool documentation and tools observed during training. Figures 4(c) and 4(d) are plotted over test instances, categorized by the tool with the lowest similarity score in the gold tool set.

ments in the Execution Failure category, demonstrating that tool calling benefits significantly from tool trials, which helps avoid incorrect tool invocations and parameter-passing errors.

5.4 Tool Use Generalization of ToolMaster

To evaluate the tool-use generalization capability of ToolMaster, we adopt ToolMaster w/o Trial-and-Exec as the baseline, which relies solely on golden tool-use trajectories during optimization.

In this experiment, we further categorize the ground-truth tools used in ToolHop into Low, Medium, and High similarity groups. For similarity computation, we employ the Qwen3-Embedding-8B model (Zhang et al., 2025) to obtain vector representations of tool documentation, and calculate similarity scores using the dot product. We then evaluate the performance of ToolMaster in terms of the calling success rate of all golden tools, and tool-calling frequency when tools from all three similarity groups are required. Finally, we show the effectiveness of ToolMaster in handling both familiar and unfamiliar questions that involve tools with varying similarity levels.

As shown in Figure 4(a), we first present the distribution of tools across different similarity levels. The results indicate that the tools are nearly uniformly distributed among the three groups, highlighting the necessity of evaluating tool use across varying similarity levels. We then report the tool-calling success rates for different similarity groups in Figure 4(b). The evaluation results show that ToolMaster consistently outperforms the baseline, with particularly notable improvements in the Low and High groups. These findings suggest that the tool trial phase enables the model to accumulate practical tool-use experience, which substantially improves tool-calling success for tools with low similarity. Moreover, the gains observed in the High similarity group indicate that ToolMaster alleviates overfitting to the golden tool-use trajectories.

Next, we randomly sample 330 test instances that require collaborative use of tools from all three similarity groups to solve the query. We report the tool-calling frequency in Figure 4(c). The results show that the baseline model prefers tools from the High similarity group, revealing an unnecessary tool-calling bias. In contrast, ToolMaster effectively mitigates this bias and exhibits a nearly uniform calling frequency across all groups. Furthermore, we categorize queries into unfamiliar (requiring at least one tool from the Low similarity group) and familiar (all required tools belong to the High similarity group) based on the golden tool set. As shown in Figure 4(d), the evaluation results show that ToolMaster yields larger performance gains on problems involving unfamiliar tools, highlighting its strong robustness and generalization ability in handling real-world tool usage scenarios.

6 Conclusion

This paper introduces ToolMaster, a novel framework that applies a Trial-and-Execution paradigm to optimize tool-augmented language models. Specifically, ToolMaster trains LLMs to imitate teacher trajectories that explicitly incorporate tool trials, where tools are invoked to obtain feedback prior to the execution phase. In addition, an RL stage is employed to further jointly coordinate trial strategies. Extensive experimental results demonstrate that ToolMaster effectively benefits from tool trial interactions, enabling models to better handle unfamiliar tools.

Limitations

Although ToolMaster demonstrates its effectiveness in improving the robustness and generalization of tool usage, the efficiency of the inference process is still constrained by the inherent nature of the trial-and-execution paradigm. Specifically, since ToolMaster relies on generating additional trial steps to proactively verify assumptions, the total inference time is inevitably constrained by the latency of receiving responses from external tools during the trial phase. Additionally, ToolMaster can be applied to diverse real-world environments containing tools with varying functionalities and shows its effectiveness. The safety of deployment may be compromised when tools that induce side effects (e.g., data modification) are involved, due to the model's autonomous tendency to explore unknown tool behaviors. This further underscores the importance of implementing rigorous safety guardrails or sandboxed environments when deploying ToolMaster in high-stakes applications.

References

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, and 9 others. 2022. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models. *Preprint*, arXiv:2012.07805.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qingguo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

DeepSeek-AI. 2024. Deepseek-v3 technical report.

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.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, and 1 others. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *ArXiv preprint*, abs/2303.16434.

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. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yifei Lu, Fanghua Ye, Jian Li, Qiang Gao, Cheng Liu, Haibo Luo, Nan Du, Xiaolong Li, and Feiliang Ren. 2025. CodeTool: Enhancing programmatic tool invocation of LLMs via process supervision. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18287–18304, Vienna, Austria. Association for Computational Linguistics.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *ArXiv preprint*, abs/2308.08747.

Zhiyuan Ma, Jiayu Liu, Xianzhen Luo, Zhenya Huang, Qingfu Zhu, and Wanxiang Che. 2025. Advancing tool-augmented large language models via meta-verification and reflection learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25. ACM.

Dheeraj Mekala, Jason Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. 2024. Toolverifier: Generalization to new tools via self-verification.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

OpenAI. 2024. GPT-4o mini: advancing cost-efficient intelligence. Accessed: 2025-12-31.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *Preprint*, arXiv:2205.12255.

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.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, and 22 others. 2023. Tool learning with foundation models.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Qwen. 2024. Qwen2.5: A party of foundation models.

Qwen. 2025. Qwen3 technical report.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models.

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. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023a. Restgpt: Connecting large language models with real-world restful apis.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023b. Restgpt: Connecting large language models with real-world restful apis.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, and 41 others. 2022. Lamda: Language models for dialog applications. *Preprint*, arXiv:2201.08239.

Hanbin Wang, Zhenghao Liu, Shuo Wang, Ganqu Cui, Ning Ding, Zhiyuan Liu, and Ge Yu. 2024. Intervenor: Prompting the coding ability of large language models with the interactive chain of repair. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2081–2107.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. 2025a. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use.

Junjie Ye, Changhao Jiang, Zhengyin Du, Yufei Xu, Xuesong Yao, Zhiheng Xi, Xiaoran Fan, Qi Zhang, Tao Gui, Xuanjing Huang, and Jiecao Chen. 2025b. Feedback-driven tool-use improvements in large language models via automated build environments.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2025. Steptool: Enhancing multi-step tool usage in llms via step-grained reinforcement learning. In *Proceedings*

of the 34th ACM International Conference on Information and Knowledge Management, CIKM '25, page 3952–3962, New York, NY, USA. Association for Computing Machinery.

Yirong Zeng, Xiao Ding, Yutai Hou, Yuxian Wang, Li Du, Juyi Dai, Qiuyang Ding, Duyu Tang, Dandan Tu, Weiwen Liu, Bing Qin, and Ting Liu. 2025. Tool zero: Training tool-augmented LLMs via pure RL from scratch. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 9135–9147, Suzhou, China. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *ArXiv preprint*, abs/2506.05176.

A Appendix

A.1 License

The licenses of the resources used in this study are as follows: StableToolBench is released under the Apache License 2.0; ToolHop is released under the CC BY 4.0 license; and RestBench-TMDB is distributed under the MIT License.

A.2 Details of Implementation

This subsection provides additional implementation details for our experimental setup.

Data Synthesis Prompt. Table 11 presents the prompt template used during data synthesis. This prompt is designed to guide the assistant to resolve user queries exclusively through tool usage. It first requires the construction of a global plan, which must explicitly include an "exploration phase" for verifying tool functionality using sample inputs before addressing the main task. Subsequently, the prompt enforces a structured execution procedure consisting of sub-goal decomposition, validation, and backtracking. Any intermediate failure must trigger strategy revision. By enforcing explicit reasoning traces and iterative problem-solving, this design enables the synthesis of high-quality interaction data that captures realistic agent behaviors, including exploratory tool testing and error recovery.

Trajectory Filter Prompt. Table 12 illustrates the prompt used to filter generated trajectories. This prompt acts as an expert evaluator that rigorously inspects conversation logs to determine whether the LLM exhibits targeted advanced problem-solving behaviors. Specifically, it enforces the presence of three mandatory components: global planning and decomposition, explicit tool exploration, and self-correction. The prompt verifies whether the agent actively attempts to "test" or understand tool functionality and whether it demonstrates resilience through self-correction upon encountering errors. By defining precise evaluation criteria, such as accepting narrative descriptions as valid plans while strictly requiring exploratory intent, we ensure that only trajectories containing high-quality autonomous reasoning patterns are retained for training.

System Prompt. Table 13 shows the system prompt template applied consistently during both training and inference to ensure behavioral alignment. This prompt standardizes the model's operational protocol by enforcing a structured reasoning workflow: first generating a global plan, then executing steps with explicit reasoning enclosed in `<think>` tags, and finally invoking tools using a predefined XML format (`<tool_call>`). By maintaining an identical prompt structure across stages, we ensure that the model internalizes the correct conventions for tool definitions (`<tools>`), intermediate reasoning, and final answer generation (`<answer>`).

ToolMaster Training Details. During training, we first apply supervised fine-tuning (SFT) followed by reinforcement learning using Group Relative Policy Optimization (GRPO). SFT is conducted for 3 epochs with a per-device batch size of 1, gradient accumulation steps of 16, a learning rate of 1×10^{-5}, and a cosine scheduler with 4% warmup. The maximum sequence length is set to 8,192 tokens. GRPO training uses a learning rate of 1×10^{-6}, a per-device batch size of 4, gradient accumulation steps of 2, and 4 generations per prompt. The KL divergence coefficient β is set to 0.002, with 2 iterations per update. The maximum prompt length is 1,024 tokens and the maximum completion length is 4,096 tokens. All training is performed using mixed precision (bf16) with gradient checkpointing enabled. All experiments are conducted on NVIDIA A800 GPUs. During testing, we fix the temperature to 0.1 and `max_tokens` to 8,192 across all models to ensure consistent evaluation.

Reward Function for GRPO Training. The reward function consists of two components: a format reward and a correctness reward. The format reward verifies whether the output adheres to the prescribed protocol, requiring reasoning to be enclosed in `<think>` tags, tool calls in `<tool_call>` tags, and the final answer in `<answer>` tags. The correctness reward is a binary score (0 or 1) assigned by a capable evaluator model, which determines whether the response fully resolves the query. The evaluation prompt is provided in Table 14.

Error Analysis Prompt. To explicitly analyze the distribution of tool-calling errors presented in Section 5.3, we employ a judge model to classify failed trajectories. Table 15 presents the prompt template used for this error type analysis. This prompt instructs the evaluator to categorize errors into three distinct types: Under-calling, Tool Execution Failure, and Reasoning Discontinuity, based on the provided taxonomy.

A.3 Details of Datasets Used in Experiments

This section provides detailed descriptions of the three benchmarks used in our evaluation: StableToolBench, TMDB, and ToolHop. These benchmarks are selected to cover a broad spectrum of tool-learning challenges, ranging from API stability and orchestration to complex multi-hop reasoning.

StableToolBench. StableToolBench is divided into six subsets based on difficulty and instruction type. I1 Instruction (I1 Inst), I1 Category (I1 Cat), and I1 Tool focus on single-tool scenarios grounded in documentation, category-level descriptions, or specific functionalities. I2 Category (I2 Cat) and I2 Instruction (I2 Inst) introduce compositional reasoning involving two distinct tools. Finally, I3 Instruction (I3 Inst) contains complex queries that require coordinating three tools. The primary evaluation metric is the Solvable Pass Rate (SoPR), computed by an evaluator model using the prompt provided in Table 14.

TMDB. The TMDB dataset (Song et al., 2023b) simulates a RESTful API environment for movie-related data and evaluates the ability to navigate complex API schemas. Tasks require answering natural language queries about movies, actors, and ratings (e.g., "Find the release date of the movie directed by X starring Y"). This benchmark involves chaining over 50 distinct API endpoints, requiring models to perform multi-step operations such as entity identification, detail retrieval, and result filtering while correctly handling interdependent parameters.

ToolHop. ToolHop (Ye et al., 2025a) emphasizes multi-hop reasoning with more than 3,000 locally executable tools, removing network latency while preserving functional complexity. Queries are inherently compositional, where the output of one tool (e.g., currency conversion) becomes the input to subsequent tools. Performance is evaluated based on both final answer accuracy against a gold standard and the validity of the execution path.

A.4 More Details of Baseline Models

To assess the effectiveness of our proposed framework, we compare it against a diverse set of baselines spanning traditional supervised fine-tuning and advanced reinforcement learning paradigms. These baselines represent prominent approaches to tool-use learning, covering both data-centric trajectory distillation and policy-centric optimization strategies.

Distill (SFT) serves as a supervised fine-tuning baseline trained on high-quality tool-use trajectories generated by DeepSeek-V3.1. These trajectories function as gold-standard execution paths, providing direct supervision from user instructions to correct API calls and final answers. This baseline primarily evaluates the model's capacity to imitate expert behavior via next-token prediction.

ToolLLM (Qin et al., 2024) is a representative data-centric framework that emphasizes high-quality instruction-tuning data construction. It employs a Depth-First Search Decision Tree (DFSDT) to explore solution spaces in complex tool-use scenarios, enabling recovery from failed attempts and identification of optimal execution paths. We re-implement ToolLLM on the Qwen2.5-7B base model to ensure a fair and up-to-date comparison.

StepTool (Yu et al., 2025) focuses on fine-grained optimization of the tool-calling process. Instead of relying solely on sparse end-of-trajectory rewards, StepTool adopts Proximal Policy Optimization (PPO) with step-wise rewards. This design enables explicit credit assignment for correct intermediate tool invocations and facilitates learning dependencies among sequential API calls.

FTRL (Ye et al., 2025b) is an RL-based approach built on the GRPO framework that leverages environmental feedback as a primary signal for policy updates. By tracing feedback loops, FTRL improves the model's ability to dynamically adjust trajectories based on execution outcomes. In this work, we re-implement FTRL using our training dataset to facilitate a fair comparison under identical experimental settings.

ToolRL (Qian et al., 2025) also adopts the GRPO framework but differentiates itself through its reward design. It optimizes the trade-off between tool-call accuracy and final answer quality by comparing groups of generated trajectories, encouraging the model to favor execution paths that are both successful and schema-compliant.

A.5 Case study

The comparison between Table 6 and Table 7 underscores the critical importance of the Trial Phase. The baseline method (ToolRL) fails due to parameter hallucination (e.g., inventing `output_format`) and an improper tool selection strategy, resulting in cascading API errors. In contrast, ToolMaster leverages the Trial Phase to first verify the functionality of the `extract_first_name` tool. This preliminary exploration validates the tool schema

Method	I1	I2	I3	Avg.
Vanilla	2.30	2.62	4.13	2.71
Distill (SFT)	4.61	4.59	7.87	5.15
ToolLLM	5.15	5.49	7.41	5.64
ToolRL	3.45	3.39	4.66	3.63
FTRL	4.35	4.12	6.56	4.64
ToolMaster	5.27	5.16	8.59	5.78

Table 4: Comparison of the average number of tool calls across different methods. All models are implemented using Qwen2.5-7B-Instruct.

Method	TMDB	ToolHop	Avg.
Vanilla	72.91	66.36	69.64
Distill (SFT)	80.17	72.56	76.37
ToolLLM	78.58	<u>75.45</u>	<u>77.02</u>
FTRL	73.92	61.21	67.57
ToolRL	<u>83.67</u>	63.20	73.44
ToolMaster	**91.25**	**80.84**	**86.05**

Table 5: Comparison of Correct Path Rates on TMDB and ToolHop benchmarks. All models are implemented using Qwen2.5-7B-Instruct.

and correctly resolves initial sub-goals (name extraction), thereby enabling accurate and error-free execution of subsequent multi-hop reasoning steps, such as identifying siblings and computing letter differences.

A.6 Cases of Tools with Different Embedding-based Similarity

To validate the rationale for using embedding similarity to distinguish between *Familiar* and *Unfamiliar* tools, we conduct a comparative analysis of three representative cases (Tables 8, 9, and 10). The progression of similarity scores illustrates the model's sensitivity in quantifying functional substitutability.

The low-similarity case (Table 8, score 0.56) demonstrates the model's ability to distinguish divergent intents (extraction vs. generation) despite shared domain keywords, correctly categorizing such tools as *Unfamiliar* to prevent misuse. The medium-similarity case (Table 9, score 0.76) highlights inferential reasoning, where a "Historical Figures" endpoint implicitly satisfies a "Family Relationship" query and is therefore classified as potentially *Familiar*. The high-similarity case (Table 10, score 0.86) confirms that explicit alignment in domain-specific terminology (e.g., "occupation", "title") yields near-synonymous interpretations. This graded spectrum of semantic alignment—from surface relevance to deep functional equivalence—provides strong empirical support for our classification strategy: high-overlap tools enable experience transfer (Exploitation), whereas lower-overlap tools necessitate new learning (Exploration).

A.7 Efficiency Analysis of ToolMaster

We further analyze inference efficiency by comparing the average number of tool invocations required to complete tasks, as reported in Table 4.

Although simpler baselines such as Zero-shot and ToolRL exhibit lower average tool usage, this typically reflects an inability to sustain the multi-step reasoning chains required for complex queries rather than genuine efficiency. Importantly, compared with the competitive baseline ToolLLM, ToolMaster exhibits a highly comparable tool usage profile with only a marginal increase in calls. This indicates that ToolMaster introduces minimal computational overhead, and the slight increase in inference cost is a reasonable trade-off for the substantial gains in task success rate and reasoning robustness.

A.8 Tool Selection Analysis

To further assess the fidelity of intermediate reasoning, we conduct an additional evaluation focusing on tool selection correctness. We employ the Correct Path Rate metric on both the TMDB and ToolHop benchmarks. This metric is defined as the recall of the tool usage trajectory, measuring the proportion of ground-truth tools correctly invoked by the model relative to the total set required to solve the query.

As shown in Table 5, ToolMaster consistently outperforms all baseline methods in tool selection accuracy. While baseline models frequently fail to identify the complete set of required tools for complex queries, our method aligns more closely with the ground-truth execution paths. These results suggest that the trial-and-execution paradigm effectively guides the model to eliminate irrelevant options and accurately identify appropriate tools, thereby establishing a stronger foundation for correct final answers.

Input Details

Query: How many additional letters does the first name of Lori Lyons have compared to the first name of Frances Freeling Broderip's sibling?
Available APIs:
`family_relationship_finder(individual_name: str, relationship_type: str, ...)`
`enhanced_family_relationship_finder(person_name: str, data_source: str, ...)`
`extract_first_name(names: array)`
`count_letters(input: str)`
`advanced_arithmetic_calculator(operation: str, numbers: array)`

Step 1: Trial Phase
Thought: <think>1. Exploration Phase: First, I need to understand the functionality of the tools. I'll start by exploring the 'extract_first_name' tool to extract the first names from the given names. I'll test with "Lori Lyons" and "Frances Freeling Broderip" to see how the tool works. 2. Sub-goal 1: Extract the first name "Lori" from "Lori Lyons"... I'll start with the exploration phase by testing the 'extract_first_name' tool on "Lori Lyons" to see how it works.</think> **Action:** <tool_call>"name": "extract_first_name", "arguments": "full_names": ["Lori Lyons"]</tool_call>" **Observation:** ""Lori""
Step 2: Trial Phase
Thought: Great! The tool correctly extracted "Lori" from "Lori Lyons". Now, I'll test it on "Frances Freeling Broderip" to extract "Frances".</think> **Action:** <tool_call>"name": "extract_first_name", "arguments": "full_names": ["Frances Freeling Broderip"]</tool_call>" **Observation:** Frances
Step 3: Execution Phase
Thought: <think>Perfect! The tool works well for both names. Now, I need to find the siblings of "Frances Freeling Broderip" using the 'family_relationship_finder' tool. I'll set the relationship type to "sibling" and use "Frances Freeling Broderip" as the individual.</think> **Action:** <tool_call>"name": "family_relationship_finder", "arguments": "individual_name": "Frances Freeling Broderip", "relationship_type": "sibling"</tool_call> **Observation:** Tom Hood
Step 4: Execution Phase
Thought: "<think>Interesting! The tool found one sibling for "Frances Freeling Broderip": "Tom Hood". Now, I need to extract the first name "Tom" from "Tom Hood" using the extract_first_name tool.</think> **Action:** <tool_call>"name": "extract_first_name", "arguments": "full_names": ["Tom Hood"]</tool_call> **Observation:** Tom
Step 5: Execution Phase
Thought: "<think>Excellent! The tool extracted "Tom" from "Tom Hood". Now, I have both first names: "Lori" (from "Lori Lyons") and "Tom" (from "Tom Hood"). I'll use the 'count_letters' tool to count the letters in each name. I'll start with "Lori".</think> **Action:** <tool_call>"name": "count_letters", "arguments": "input": "Lori"</tool_call>" **Observation:** 4
Step 6: Execution Phase
Thought: <think>Great! "Lori" has 4 letters. Now, I'll count the letters in "Tom".</think> **Action:** <tool_call>"name": "count_letters", "arguments": "input": "Tom"</tool_call> **Observation:** 3
Step 7: Final Response
Thought:<think>Perfect! "Tom" has 3 letters. The final step is to calculate the difference: 4 (Lori) - 3 (Tom) = 1. So, "Lori" has 1 more letter than "Tom". I'll validate all steps: we successfully extracted the first names, found the sibling, and counted the letters. The answer is 1.</think> **Final Answer:** <answer>The first name "Lori" has 1 additional letter compared to the first name "Tom" of Frances Freeling Broderip's sibling.</answer>

Table 6: Case study 1: The correct solution path generated by ToolMaster.

Input Details

Query: How many additional letters does the first name of Lori Lyons have compared to the first name of Frances Freeling Broderip's sibling?
Available APIs:
`family_relationship_finder(individual_name: str, relationship_type: str, ...)`
`enhanced_family_relationship_finder(person_name: str, data_source: str, ...)`
`extract_first_name(names: array)`
`count_letters(input: str)`
`advanced_arithmetic_calculator(operation: str, numbers: array)`

Step 1: Execution Phase
Thought: <think>I need to find the first names of Lori Lyons and Frances Freeling Broderip's sibling using the family relationship finder tools. First, I'll start by finding the first name of Lori Lyons using the family relationship finder.</think> **Action:** <tool_call>{"name": "family_relationship_finder", "arguments": {"individual_name": "Lori Lyons", "relationship_type": "child", "data_source": "FamilySearch", "output_format": "json"}}</tool_call> **Observation:** "an error occured when call family_relationship_finder: family_relationship_finder() got an unexpected keyword argument 'output_format'"
Step 2: Execution Phase
Thought: <think>I received an error because I specified "output_format" which is not a required or optional parameter for the family_relationship_finder. I'll try without it.</think> **Action:** <tool_call>{"name": "family_relationship_finder", "arguments": {"individual_name": "Lori Lyons", "relationship_type": "child", "data_source": "FamilySearch"}}</tool_call> **Observation:** "Error: No data found for Lori Lyons with relationship type child."
Step 3: Execution Phase
Thought: <think>I got an error for Lori Lyons. Let me try with a different relationship type, like "parent", to see if that works better.</think> **Action:** <tool_call>{"name": "family_relationship_finder", "arguments": {"individual_name": "Lori Lyons", "relationship_type": "parent", "data_source": "FamilySearch"}}</tool_call> **Observation:** "Error: 'relationship_type' must be one of ['father', 'mother', 'sibling', 'child', 'grandparent', 'aunt', 'uncle', 'cousin']."
Step 4: Execution Phase
Thought: <think>I see that "parent" is not a valid relationship type for this tool. The valid types are listed in the enum. Since "child" and "parent" are not working, I'll try "sibling" as it might be relevant for finding first names. ... I'll use the enhanced family relationship finder for this, as it might handle the dataset aspect better.</think> **Action:** <tool_call>{"name": "enhanced_family_relationship_finder", "arguments": {"person_name": "Frances Freeling Broderip", "data_source": "FamilySearch", "relationship_type": "sibling", "output_format": "json"}}</tool_call> **Observation:** "an error occured when call enhanced_family_relationship_finder: enhanced_family_relationship_finder() got an unexpected keyword argument 'output_format'"
Step 5: Execution Phase
Thought: <think>I received an error for the enhanced family relationship finder as well, due to the "output_format" parameter. I'll try without it.</think> **Action:** <tool_call>{"name": "enhanced_family_relationship_finder", "arguments": {"person_name": "Frances Freeling Broderip", "data_source": "FamilySearch", "relationship_type": "sibling"}}</tool_call> **Observation:** {"error": "Person not found in the data source."}
Step 6: Final Response
Thought: <think>I have attempted to retrieve the first names using the available tools, but both queries resulted in errors or data not found. ... Since I've tried the available approaches and encountered limitations, I should provide a final answer based on the findings.</think> **Final Answer:** <answer>I attempted to retrieve the first names using the available tools, but encountered several limitations. The family relationship finder for "Lori Lyons" resulted in an error when specifying an "output_format," and when I removed it, it returned "No data found for Lori Lyons with relationship type child." The enhanced family relationship finder for "Frances Freeling Broderip" also had an issue with the "sibling" relationship type, returning an error. Based on the tool descriptions, these tools are designed for finding familial relationships rather than extracting first names from names. ... Due to these limitations, I was unable to retrieve the specific first names for the calculation.</answer>

Table 7: Case study 1: The failed solution path generated by ToolRL.

Tool Pair Comparison (Calculated Similarity: 0.5616)

Tool A: `extract_first_letter`

Description: An enhanced tool designed to extract the first letter from a given string. It supports various string types and includes options for handling edge cases.

Parameters:

- **Required:** `input` (the source string)
- **Optional:** `ignore_whitespace`, `ignore_non_alphabetic`, `character_set`, `return_format`

Tool B: `random_word_api_get_word_by_start_and_contain`

Description: Returns a random word that starts with a certain string and contains a certain string (e.g., starts with "fru", contains "le").

Parameters:

- **Required:** `start` (prefix string), `substring` (contained string)

Template Response: `word` (string)

Analysis:

The model assigns a moderate similarity score (**0.5616**), reflecting a nuanced understanding of the functional overlap without over-matching.

- **Shared Context:** Both tools operate heavily within the domain of *string manipulation* and *character positioning*. Tool A focuses on the "first letter" (a specific position), while Tool B filters words based on how they "start" (a positional constraint).
- **Distinct Intent:** The score is in low similarity groups because the core intents differ: Tool A is an *extraction* utility (deterministic processing), whereas Tool B is a *generation/retrieval* API (randomized output). The embedding correctly identifies them as related "word/string tools" but distinct enough to avoid confusion in high-precision retrieval tasks.

Table 8: Case study 2: Analysis of low semantic similarity tools.

Tool Pair Comparison (Calculated Similarity: 0.7586)

Tool A: `family_relationship_finder`

Description: An advanced tool designed to identify a variety of family relationships, including parentage, siblings, and offspring, for historical and contemporary figures.

Parameters:

- **Required:** `person_name`, `relationship_type` (enum: father, mother, sibling, child)
- **Optional:** `historical_context`, `data_source_preference`, `date_range`

Tool B: `historical_figures_by_api_ninjas_v1_historicalfigures`

Description: API Ninjas Historical Figures API endpoint. Returns a list of up to 10 people that match the search parameters.

Parameters:

- **Optional:** `name` (e.g., "julius caesar"), `offset`

Key Response Fields: `parents`, `children`, `spouses`, `born`, `died`

Analysis:

Despite differences in naming conventions and parameter structures, the model assigns a high similarity score (**0.7586**).

- **Parameter Mapping:** The model detects semantic equivalence between Tool A's `person_name` and Tool B's `name`. Although Tool A requires specific relationship types as inputs, Tool B implicitly covers these relationships in its output schema.
- **Semantic Alignment:** The embedding successfully bridges the gap between a function-oriented tool ("Finder") and a data-oriented endpoint ("Historical Figures"). It understands that retrieving a historical figure's profile (Tool B) is semantically aligned with querying their family tree (Tool A), as evidenced by the shared concepts of "parents" and "children" in their definitions.

Table 9: Case study 3: Analysis of medium semantic similarity tools.

Tool Pair Comparison (Calculated Similarity: 0.8551)

Tool A: `genealogy_lookup`

Description: An advanced tool for retrieving detailed genealogical information about *historical figures*. Offers functionalities to explore family trees and manage titles.

Parameters:

- **Required:** `name`, `relationship` (enum: father, mother, spouse, etc.)
- **Optional:** `title`, `occupation`, `historical_context`, `geographical_location`

Tool B: `historical_figures_by_api_ninjas_v1_historicalfigures`

Description: API Ninjas Historical Figures API endpoint. Returns a list of people matching search parameters.

Parameters:

- **Optional:** `name`, `offset`

Key Response Fields: `title`, `occupation`, `parents`, `children`, `spouses`, `awards`

Analysis:

The model assigns a very high similarity score (**0.8551**), indicating a near-synonymous functional relationship.

- **Strong Domain Identity:** Unlike the previous case, Tool A explicitly includes "historical figures" in its description, creating a direct lexical and semantic match with Tool B's name and domain.
- **Comprehensive Field Mapping:** The embedding detects a dense alignment between Tool A's input parameters and Tool B's output schema.
  - Tool A's optional inputs `title` and `occupation` directly map to Tool B's response fields `title` and `occupation`.
  - Tool A's `relationship` parameter (father, spouse) perfectly corresponds to Tool B's structure (`parents`, `spouses`).

Table 10: Case study 4: Analysis of high semantic similarity tools.

Data Synthesis Prompt

Task Description

You are a smart assistant capable of utilizing provided tools to answer users' questions. Your primary strategy should be to use the available tools rather than relying on internal knowledge. Follow this enhanced process to solve problems, incorporating the behaviors of backtracking, setting sub-goals, validation, and exploration.

1. Create a global plan for the query. This plan must include:An exploration phase to understand tool functionality by making initial tool calls with sample inputs.Decomposition of the task into sub - goals when necessary.

2.Execute each step in the plan:

- Record your thought process, formatted as <thought></thought>.
- Call the appropriate tool by providing its name and parameters in JSON format, like '<|tool calls begin|><|tool call begin|>tool's name<|tool sep|>{{"parameters1": "value1", "parameters2": "value2"}}<|tool call end|><|tool calls end|>', formatted as '<|tool calls begin|><|tool call begin|><|tool call end|><|tool calls end|>'.
- Every turn must include one tool call and do not combine multiple tool calls in one turn.
- Validate whether the result meets the requirements of the current step. If not, backtrack by revising the plan and repeating the relevant steps.
- If a tool is unavailable or returns an error, consider using alternative tools that can achieve similar results, and adjust the plan accordingly.
- Try your best to use the tools to obtain information and you can call the tools multiple times if necessary.

Stopping Criteria

You can stop the problem - solving process when:

- You have obtained a result that fully satisfies the user's request.
- You have validated that the result meets all requirements.
- All sub-goals have been successfully addressed.

Tool Usage Policy

- Always prioritize using the available tools to obtain information.
- When uncertain about the accuracy or sufficiency of information from tools, perform additional tool calls or validation steps.
- Do not use internal knowledge and only rely on tools to get information.
- If a tool is unavailable, look for alternative tools within the provided documentation that can serve as substitutes to achieve the same objective.
- Include an exploration phase in your plan to better understand tool behavior before applying them to the task. This phase is mandatory and must be completed before proceeding with the main task.

Tool Documentation

Below is the documentation for available tools: {tool doc}.

Table 11: Data synthesis prompt template.

Trajectory Filter Prompt

You are an expert evaluator of AI Agent reasoning and tool usage. Your task is to analyze a conversation log between a User and an AI Assistant to determine if the Assistant exhibits a specific set of advanced problem-solving behaviors.

You must look for the presence of three distinct behaviors. The Assistant does not need to use exact keywords (like "Global Plan" or "Backtracking"), but the reasoning process in the <think> tags must clearly demonstrate these actions took place.

The Three Required Behaviors:

1. Global Planning & Decomposition:

The Assistant must set a high-level strategy at the beginning.

It should break complex user queries into smaller, manageable sub-goals or steps.

Criteria: Does the Assistant explicitly map out what it intends to do before jumping into tool calls?

2. Tool Exploration (Mandatory):

The Assistant must demonstrate an intent to "understand" or "test" a tool before fully relying on it for the final answer.

This can appear as:

Calling a tool to see its output format (schema exploration).

Calling a tool with sample data to verify behavior.

Explicitly stating in the thought process that a call is being made to "explore," "check capabilities," or "understand the response" (even if using real user data).

Criteria: Is there a step where the Agent tries to learn about the tool's behavior rather than just assuming it works perfectly immediately?

3. Validation & Backtracking (Self-Correction):

Validation: After receiving a tool output, the Assistant must evaluate if the data satisfy the user's request.

Resilience/Backtracking:

If an error occurs: The Assistant must acknowledge the error and propose a fix, a retry with different parameters, or a substitute tool.

If successful: The Assistant validates the data is correct. (Note: If the tool works perfectly, "backtracking" is not required, only validation is required).

Criteria: Does the Agent verify the results? If things go wrong, does it try to fix them instead of giving up or ignoring the error?

Evaluation Rules:

Be Lenient on Format: Do not demand specific XML tags or numbered lists for the plan. Narrative paragraphs are acceptable if the logic is present.

Contextual Exploration: "Exploration" is valid even if the agent uses the user's actual input, provided the intent described in the thought process is to verify how the tool functions or returns data.

Partial Trajectories: If the log ends abruptly (e.g., during a retry), judge based on the behaviors exhibited so far. If the agent demonstrated the intent to fix an error, that counts as satisfying the Validation/Backtracking requirement.

Output Format:

Analysis: Briefly describe where you found evidence (or lack thereof) for each of the three behaviors.

Result: Output only True if ALL three behaviors are present. Output False if ANY of the three are missing.

Table 12: Trajectory filter prompt template.

System Prompt
You are a smart assistant capable of utilizing provided tools to answer users' questions. Follow this process to solve problems: 1. Create a global plan for the query. 2. Execute each step in the plan: - Record your thought process, formatted as "<think></think>". - Call the appropriate tool by providing its name and parameters in JSON format, For each function call, return a json object with function name and arguments within <tool_call> </tool_call> XML tags:<tool_call>"name": <function-name>, "arguments": <args-json-object></tool_call> - Get the result returned by the tool, formatted as "<tool_response></tool_response>". 3. After completing all steps, provide the final answer, formatted as "<answer></answer>". You are provided with function signatures within <tools></tools> XML tags: <tools> {tool-docs} </tools>

Table 13: System prompt template.

Pass Rate Evaluation Prompt
You are an assistant responsible for evaluating whether an LLM agent's response should be counted as Pass, Fail, or Unsure in passrate calculations. Your evaluation must consider both the final answer and the complete execution chain. **Status Determination Rules:** **Pass:** Answer sufficiently solves query; Execution chain shows successful API calls; Initial errors were corrected; Information verifiable through API responses **Fail:** API observations show execution errors; Answer contradicts evidence; Information incorrect/invalid; Solution misses core requirements **Unsure:** Cannot verify authenticity; Insufficient validation data; Need complete reasoning process; **Output Format:** `{"content": "Evaluation reasoning", "answer_status": "Pass/Fail/Unsure"}` **Required Input:** Original query; Final answer; Complete execution chain with API responses

Table 14: Pass Rate evaluation prompt template followed by ToolMVR (Ma et al., 2025).

Error Type Judge Prompt template

You are an expert analyst of Large Language Model (LLM) Agent behaviors. Your task is to analyze failed execution trajectories and classify the **PRIMARY** error type into EXACTLY one of the following 3 categories.

Input Data - 'ground_truth': The correct answer. - 'response': The trajectory of thoughts, tool calls, and tool outputs.
Classification Taxonomy Please identify the **Root Cause** or the most **Fatal Error** that led to the failure. Evaluate which error type is the dominant factor.

I. Under-calling & Scope Insufficiency
* **Definition**: The agent fails to **initiate** the necessary tool calls to cover the full scope of the question. It makes incomplete attempts.
* **Scope**: The error is about **"What was NOT called"**.
* **Key Indicators**:
* **Direct Answering**: The agent answers the question directly (often hallucinating) WITHOUT calling any tools.
* **Partial Coverage**: The user asks for "Director AND Actor", but the agent ONLY calls a tool for "Director" (Missing scope).
* **Phantom Usage**: The agent claims to use a tool in thought, but no actual tool call is generated.

II. Tool Execution Failure
* **Definition**: The agent attempts to use tools, but the **Tool Layer fails to provide usable data**, and the agent fails to recover. This covers both **Technical Failures** (Syntax) and **Data Failures** (Empty Results).
* **Scope**: The error is about **"The Tool Call yielded nothing useful"**. * **Key Indicators**:
* **Syntax/Schema Errors**: Persistent JSON errors, missing parameters, or wrong types that prevent execution.
* **Empty/Null Results**: The tool runs successfully but returns "Not Found", "[]", or "None", and the agent **cannot recover** (e.g., stops, loops, or gives up).
* **Unrecovered Mechanical Loop**: Repeatedly making the exact same failed call (Syntax or Empty) without changing strategy.

III. Reasoning Discontinuity
* **Definition**: **Reasoning Process** breaks down. The logic connection between steps is flawed or incoherent.
* **Scope**: The error is about **Reasoning Discontinuity**.
* **Key Indicators**:
* **Context Loss / Interruption**: The agent starts a reasoning chain but abruptly stops or forgets previous constraints.
* **Logical Errors**: The agent forgets to the original question constraints, mixes up variables, or makes invalid deductions.

—
Guidance on Identifying the "Primary" Error
Use the following priority logic to decide the Primary one:
1. Check for Under-calling (Type I):
* Did the agent **fail to call** the necessary tools entirely?
* Did it miss a part of the question (e.g., checked date but missed location)?
* **If YES**, categorize as **I**.
2. Check for Tool/Retrieval Failure (Type II):
* Did the agent call the tool, but the tool **failed to work** (Syntax Error) OR **failed to return data** (Empty Result) and cannot recover?
* Did this failure cause the agent to get stuck, loop, or fail to produce an answer?
* **If all YES**, categorize as **II**.
3. Check for Reasoning Discontinuity (Type III):
* If agent dive in to reasoning but got lost or looped, mixed variables, or made illogical jumps?
* **If YES**, categorize as **III**.

Output Format
{ "trajectory_id": "ID or Summary", "category_code": "Category ID (I, II, or III)", "category_name": "Full Category Name", "reasoning": "Explain why this is the primary error type." } Now, analyze the following trajectory:
Question: {question}
Ground Truth: {ground_truth}
Response Trajectory: {response}

Table 15: Prompt template for error type classification.