Feedback-Driven Tool-Use Improvements in Large Language Models via Automated Build Environments

Anonymous ACL submission

Abstract

Effective tool use is essential for large language models (LLMs) to interact with their environment. However, progress is limited by the lack of efficient reinforcement learning (RL) frameworks specifically designed for tool use, due to challenges in constructing stable training environments and designing verifiable reward mechanisms. To address this, we propose an automated environment construction pipeline, incorporating scenario decomposition, document generation, function integration, complexity scaling, and localized deployment. This enables the creation of high-quality training environments that provide detailed and measurable feedback without relying on external tools. Additionally, we introduce a verifiable reward mechanism that evaluates both the precision of tool use and the completeness of task execution. When combined with trajectory data collected from the constructed environments, this mechanism integrates seamlessly with standard RL algorithms to facilitate feedback-driven model training. Experiments on LLMs of varying scales demonstrate that our approach significantly enhances the models' tool-use performance without degrading their general capabilities. Our analysis suggests that these gains result from improved context understanding and reasoning, driven by updates to the lower-layer MLP parameters in models.[1]

1 Introduction

Tool use in large language models (LLMs) (Bai et al., 2022; OpenAI, 2023; Reid et al., 2024) refers to their ability to interact with the external world by invoking tools to retrieve information (Tang et al., 2023) or respond to environmental stimuli (Ye et al., 2024a). Consequently, the effectiveness of an LLM's tool-use capabilities significantly impacts its performance on complex real-world tasks (Qin et al., 2023) and represents a key step toward achieving general-purpose AI (Xi et al., 2023).

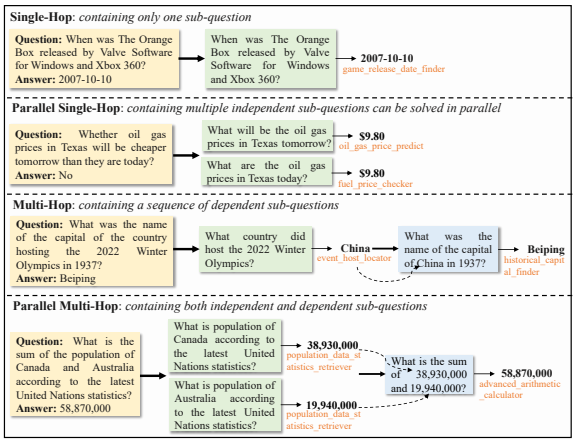

Figure 1: Illustrative examples of four scenarios, categorized by varying sub-question pattern combinations.

Recent research has focused on improving LLM tool-use capabilities through post-training methods (Qu et al., 2025). A common approach involves fine-tuning open-source models on interaction trajectories generated by proprietary LLMs with diverse APIs (Schick et al., 2023; Tang et al., 2023). To increase the diversity and quality of these trajectories, various techniques have been proposed, including Monte Carlo sampling (Yuan et al., 2025), graph-based inference (Liu et al., 2024), and other optimization strategies (Zhuang et al., 2024). In parallel, reinforcement learning (RL) methods have been explored to improve adaptability and decision-making through active environment interaction (Yu et al., 2024; Ye et al., 2024c; Qian et al., 2025).

However, current RL-based frameworks for training tool-use face significant limitations that hinder the development of robust models. One major challenge is the difficulty in constructing stable training environments. These frameworks typically rely on a large set of online tools, which are prone to service disruptions due to factors

[1]Code and data will be released upon paper acceptance.

like API rate limits (Song et al., 2023; Zhuang et al., 2023). Additionally, standardizing the organization and deployment of these tools imposes high infrastructure costs (Qin et al., 2024). Another key limitation is the lack of verifiable reward signals. The complexity of tool interactions and the diversity of valid trajectories often necessitate the use of high-level LLMs for evaluation, which introduces model bias and undermines both training efficiency and algorithmic stability (Yu et al., 2024; Qian et al., 2025).

To address these limitations, we propose an automated pipeline for generating a large number of high-quality tool-use training environments. This pipeline includes scenario decomposition, document generation, function integration, complexity scaling, and localized deployment. By executing all tools locally as code, it enables the dynamic creation of diverse, stable environments while ensuring controlled feedback, free from reliance on external online toolsets. Furthermore, each environment is built around a well-defined set of target sub-questions, allowing for precise evaluation of model behavior.

Building on this environment, we introduce a verifiable reward mechanism that evaluates the precision of tool use and the completeness of task execution solely by analyzing feedback from the environment. When combined with interaction trajectory data collected, this mechanism can be seamlessly integrated into any preference-based optimization method. Together, these components form a feedback-driven framework for training LLMs in robust and effective tool use.

We validate our approach through extensive experiments on LLMs of varying sizes. Results show that our method consistently improves tool-use performance across four distinct benchmarks, while preserving the models' general capabilities, regardless of the model architecture, inference patterns, or training algorithms employed. A parameter-level analysis suggests that these gains are largely driven by updates in lower-layer MLP parameters, which enhance the model's contextual understanding and decision-making.

In summary, our main contributions are: 1) We propose an automated environment construction pipeline that ensures stable and verifiable tool-use training; 2) We introduce a verifiable reward mechanism that jointly evaluates precision and completeness, compatible with preference-based optimization strategies; and 3) We conduct extensive experiments demonstrating substantial improvements in tool use, supported by parameter-level analysis.

2 Related Work

Methods for Designing Tool-Use Environments The construction of diverse tool-use environments is both necessary and critical for advancing research. Early studies primarily focused on a limited set of predefined tools, for which large volumes of data were either manually curated or synthetically generated within narrowly defined scenarios (Schick et al., 2023; Huang et al., 2024; Yang et al., 2023). To support more diverse tool-use tasks, later efforts collected various existing APIs from the web and organized them into large toolsets. Synthetic user inputs were then generated to evaluate the performance of LLMs (Qin et al., 2024; Song et al., 2023; Ye et al., 2025c; Zhuang et al., 2024). However, these approaches remain constrained by the quality and coverage of the collected toolsets and often suffer from issues such as unstable execution environments and unreliable feedback. To address these limitations, Ye et al. (2025a) introduced a controllable method for constructing multi-hop tool-use environments. While this represented progress, their approach was restricted to the multi-hop scenario and focused solely on building test data. In contrast, we propose a robust automated framework that supports the generation of diverse tool-use scenarios and enables both effective training and evaluation, thereby providing a more comprehensive foundation for research in this domain.

Techniques for Enhancing Tool-Use Abilities To improve the tool-use capabilities of LLMs, early research typically relied on a limited number of predefined tool environments. These efforts either used in-context learning to prompt the model to invoke tools (Paranjape et al., 2023) or manually constructed large datasets of tool-use instances for supervised training (Hsieh et al., 2023). To further enhance tool comprehension, some approaches directly encoded tools as part of the model's vocabulary during training (Hao et al., 2023; Wang et al., 2025). To encourage generalization to out-of-domain tools, many existing studies format tool document using a unified JSON Schema and train models to perform tool calls using structured formats such as ReAct (Yao et al., 2023) or

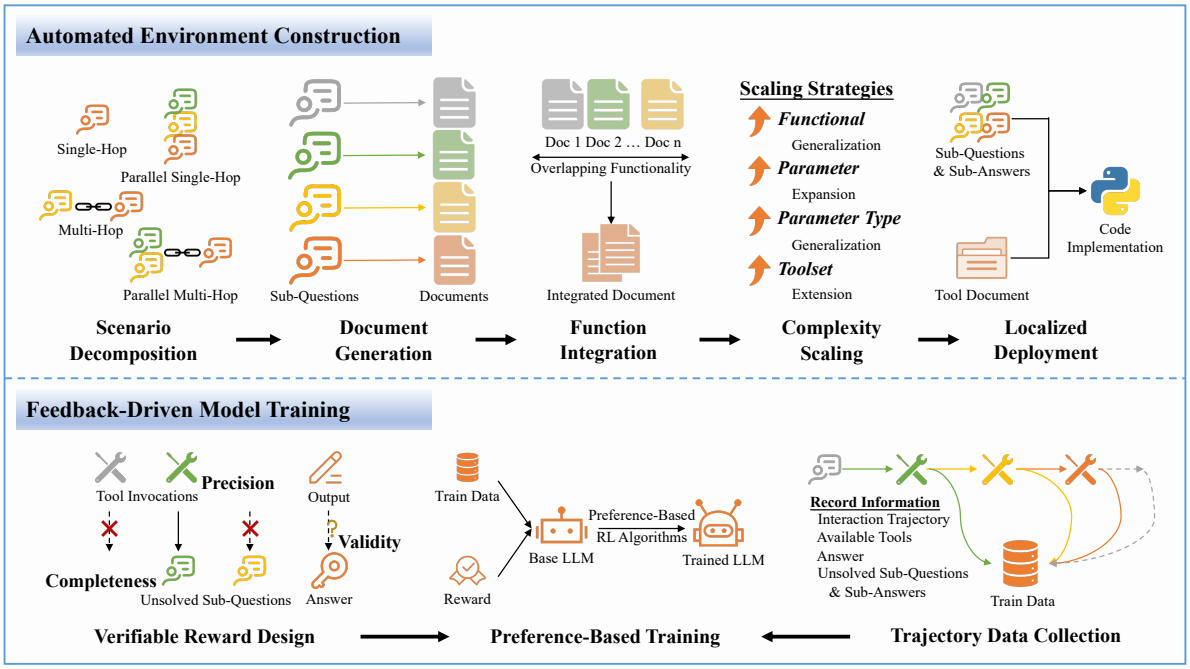

Figure 2: Overview of FTRL. The automated environment construction follows a five-stage pipeline to generate diverse tool-use training environments. Feedback-driven model training then collects data within these environments, incorporates verifiable reward mechanisms, and optimizes performance using preference-based RL algorithms.

CodeAct (Trivedi et al., 2024). With advances in RL, more recent work has explored enabling models to improve tool-use capabilities through interaction with their environments (Yu et al., 2024; Ye et al., 2024c; Qian et al., 2025). However, these approaches have yet to deliver a universal training framework, primarily due to the difficulty of efficiently constructing diverse and stable training environments, and the lack of reliable reward signals for guiding learning. In contrast, we introduce a feedback-driven training scheme grounded in the automated construction of diverse tool-use environments. Combined with a verifiable reward mechanism, our approach effectively enhances the tool-use capabilities of LLMs using only feedback from a stable environment.

3 Approaches

As illustrated in Figure 2, our approach comprises two core components: 1) a five-stage automated pipeline for constructing tool-use training environments for LLMs; and 2) a feedback-driven model training framework that leverages these environments to enhance tool-use capabilities.

3.1 Automated Environment Construction

A complete training environment for tool use consists of a user input $\mathbb{U}$ and a toolset $\mathbb{T}$. $\mathbb{U}$ contains a question q and its corresponding answer a, where q can be decomposed into a sequence of sub-questions $q_1, q_2, \ldots, q_n$. Each sub-question q_i is solvable via a tool from $\mathbb{T}$, producing intermediate answers a_i. Each tool $t_i \in \mathbb{T}$ is specified by a document d_i and an implementation component c_i. The document d_i defines the tool's name n_i, function description f_i, and a set of parameters $\mathbb{P}_i$. Each parameter $p_{ij} \in \mathbb{P}_i$ is defined by its name pn_{ij}, type pt_{ij}, description pf_{ij}, and a binary flag pr_{ij} indicating whether it is required. To efficiently construct robust training environments for tool use, we propose a five-stage automated environment construction pipeline.

Scenario Decomposition To ensure the diversity of training environments, we account for different logical relationships among sub-questions, defining four scenarios of tool use, as illustrated in Figure 1: 1) Single-hop, containing only one sub-question q_1; 2) Parallel single-hop, where q is decomposed into multiple independent sub-questions $q_1, q_2, \ldots, q_n$ that can be solved in parallel; 3) Multi-hop, where q is broken down into a sequence of dependent sub-questions, such that each q_{i+1} relies on the answer a_i of the preceding sub-question; and 4) Parallel multi-hop, a hybrid structure consisting of both independent and dependent sub-questions. We manually construct user inputs for each scenario to ensure broad coverage and task variability.

Datasets	Train	Test			
	Ours	Ours	ToolHop	τ-bench	RoTBench
# Scenarios	4	4	3	2	5
# Number	2215	200	2985	165	840
# Avg Tools	9.39	9.26	3.93	12.36	8.50

Table 1: Statistics of datasets. '# Scenarios' refers to the number of tool-use scenarios; '# Number' indicates the total number of data instances; '# Avg Tools' denotes the average number of tools included in each instance.

Document Generation After constructing diverse user inputs, we ensure their solvability by generating a corresponding tool document d_i for every sub-question q_i. For every sub-question q_i, we generate a tool document d_i with a function f_i explicitly designed to solve the task described by q_i. The parameter set $\mathbb{P}_i$ is also abstracted directly from q_i, establishing a precise one-to-one mapping between sub-questions and tool interfaces.

Function Integration While generating a separate tool for each sub-question ensures full coverage, it often results in redundancy within the toolset. To reduce duplication, we analyze tool documents $d_1, d_2, \ldots, d_n$ and merge those with overlapping functionality. The resulting set $d_1, d_2, \ldots, d_m$ $(m \leq n)$ offers improved modularity and efficiency, while preserving logical consistency with the original task.

Complexity Scaling While integrating functionality leads to a more organized toolset, the resulting tools often exhibit relatively simple functionality and limited parameter sets (Ye et al., 2025a), which can constrain the model's ability to generalize to more complex tools. To address this limitation, we enhance tool complexity through four key strategies: 1) Functional generalization: Expand each tool's function f_i beyond its original sub-questions to cover a broader range of tasks, thereby increasing its versatility and data processing capability; 2) Parameter expansion: Enrich the parameter set $\mathbb{P}_i$ to support the extended functionality; 3) Parameter type generalization: Replace simple data types (e.g., strings) with more complex structures such as arrays, dictionaries, and nested types; and 4) Toolset extension: Introduce additional non-essential tools to increase the complexity of tool selection and elevate the overall difficulty of the environment. These enhancements result in a more realistic and challenging setting for tool use, better suited for training models capable of robust generalization.

Localized Deployment Once the tool document d_i is finalized, it is mapped to a corresponding Python function and deployed locally. Key elements such as n_i, pn_{ij}, and pt_{ij} are mapped to function name, arguments, and type annotations, respectively. Additionally, the associated q_i and a_i are used as prior conditions to ensure that the function returns correct outputs when invoked with valid arguments. This design establishes a stable and controlled environment, facilitating the effective learning of tool use through interaction.

Through these five stages, we are able to construct complete tool-use training environments without relying on external toolsets, significantly enhancing the scalability of training data. Since all tools are locally deployable, the environment can provide stable and consistent feedback. Moreover, by analyzing this feedback, we can accurately evaluate each step of the tool invocation process.

3.2 Feedback-Driven Model Training

Building on the constructed training environments, we propose a feedback-driven training framework. By designing verifiable reward mechanisms that rely solely on environment feedback, it support various preference-based training strategies. This allows the model to continually improve its tool-use capabilities through interaction.

Verifiable Reward Design To enable continuous improvement, accurate and informative reward signals are essential (Liu et al., 2025). Leveraging stable environments we construct, the model receives direct feedback after each tool invocation, which allows us to verify whether a sub-question has been successfully solved. This enables the generation of step-level, verifiable rewards. To encourage both correctness and efficiency, we draw inspiration from the F1 score (Chinchor, 1992), balancing the precision of tool invocation and the completeness of task execution. We also assess the validity of the model's final answer, resulting in a comprehensive reward signal. Importantly, the framework operates without external models or predefined solution paths, enhancing generality and applicability. Specially, let o be the model's output, p be the number of tool invocations, $q \leq p$ be the number of sub-questions successfully solved, $t \geq 0$ be the number of remaining unsolved sub-questions, and a be the correct final answer in the

Family	Version	Ours			ToolHop	τ-bench	RoTBench			Avg
		Solve-P	Solve-R	Solve-F1	AC	Pass^1	TS	PI	CF	
Closed-Source LLMs										
Gemini	2.5-Flash	**52.78**	23.13	23.24	39.37	40.04	53.62	34.95	22.76	36.24
Gemini	2.5-Pro	51.77	31.96	29.98	45.32	47.09	49.05	42.38	27.52	40.63
Claude	4.0-Sonnet	42.33	56.07	42.59	**53.97**	**50.22**	39.72	34.95	21.81	42.71
GPT	3.5-Turbo	25.77	29.29	25.66	25.73	15.13	65.71	29.81	17.33	29.30
GPT	4o	33.43	42.01	33.39	36.31	37.43	**78.76**	**49.43**	**31.52**	42.79
Open-Source LLMs (32B - 72B)										
Qwen2.5-32B	Instruct	30.12	37.80	32.42	19.70	21.91	75.43	36.86	20.00	34.28
Qwen2.5-72B	Instruct	31.68	43.41	35.22	30.29	34.26	72.19	26.29	17.14	36.31
Qwen3-32B	Non-Reasoning	33.20	46.24	36.34	35.34	27.39	73.24	41.71	24.86	39.79
Qwen3-32B	Reasoning	28.94	40.41	31.30	50.12	31.00	53.90	33.81	20.19	36.21
Open-Source LLMs (7B - 14B)										
Qwen2.5-7B	Instruct	27.44	27.69	25.97	11.99	5.91	70.38	27.33	15.43	26.52
	FTRL-Reinforce++	41.61 ↑	41.36 ↑	40.36 ↑	23.45 ↑	11.91 ↑	70.95 ↑	40.95 ↑	26.10 ↑	37.09 ↑
	FTRL-GRPO	47.82 ↑	47.37 ↑	46.78 ↑	29.68 ↑	6.91 ↑	74.00 ↑	30.86 ↑	18.95 ↑	37.80 ↑
Qwen2.5-14B	Instruct	33.31	34.64	32.76	25.80	16.74	71.81	38.29	21.33	34.33
	FTRL-Reinforce++	49.51 ↑	47.20 ↑	47.57 ↑	36.42 ↑	26.83 ↑	73.05 ↑	45.24 ↑	28.19 ↑	44.25 ↑
	FTRL-GRPO	48.67 ↑	50.56 ↑	49.20 ↑	26.63 ↑	25.43 ↑	71.52 ↓	36.19 ↓	21.62 ↑	41.23 ↑
Qwen3-8B	Non-Reasoning	21.18	30.71	23.48	28.54	10.13	75.52	36.29	22.19	31.01
	FTRL-Reinforce++	41.96 ↑	47.37 ↑	43.63 ↑	37.79 ↑	21.96 ↑	78.10 ↑	42.57 ↑	25.90 ↑	42.41 ↑
	FTRL-GRPO	46.44 ↑	55.84 ↑	49.54 ↑	38.93 ↑	23.35 ↑	77.62 ↑	44.67 ↑	27.05 ↑	**45.43** ↑
Qwen3-14B	Non-Reasoning	28.29	37.58	29.97	24.19	13.74	75.90	36.38	20.67	33.34
	FTRL-Reinforce++	44.06 ↑	**59.30** ↑	48.24 ↑	38.32 ↑	17.61 ↑	77.81 ↑	42.57 ↑	25.24 ↑	44.14 ↑
	FTRL-GRPO	49.41 ↑	56.71 ↑	**51.82** ↑	38.26 ↑	24.26 ↑	75.33 ↓	40.95 ↑	22.48 ↑	44.90 ↑
Qwen3-8B	Reasoning	27.78	38.38	29.52	40.70	16.43	54.00	34.29	20.38	32.68
	FTRL-Reinforce++	38.11 ↑	43.70 ↑	39.40 ↑	41.24 ↑	32.52 ↑	53.52 ↓	37.52 ↑	22.29 ↑	38.54 ↑
	FTRL-GRPO	40.67 ↑	40.06 ↑	38.58 ↑	40.50 ↓	28.91 ↑	54.57 ↑	39.24 ↑	22.95 ↑	38.19 ↑
Qwen3-14B	Reasoning	31.91	40.23	32.96	40.10	18.87	57.33	36.19	20.29	34.74
	FTRL-Reinforce++	44.15 ↑	48.11 ↑	44.04 ↑	44.79 ↑	27.09 ↑	58.95 ↑	37.24 ↑	21.62 ↑	40.75 ↑
	FTRL-GRPO	41.67 ↑	47.67 ↑	43.36 ↑	41.64 ↑	31.70 ↑	60.57 ↑	38.10 ↑	21.43 ↑	40.77 ↑

Table 2: Performance of different LLMs on each test set. The best result for each dataset is **bolded**, and the second-best is underlined. Performance improvements over the base model after training with our method (i.e., FTRL-Reinforce++ and FTRL-GRPO) are indicated with ↑, while performance declines are marked with ↓.

environment. The reward R is defined as:

$$R = \begin{cases} \frac{2q}{p+1}, & \text{if } p > 0 \\ -0.5, & \text{elif } o \text{ is None} \\ -0.3, & \text{elif there is a format error} \\ \frac{1}{t+1}, & \text{elif } a \text{ is in } o \\ 0.5, & \text{elif } t = 0 \\ 0, & \text{else} \end{cases}$$

Trajectory Data Collection Suppose we have an LLM $\mathcal{M}$ to be optimized, leveraging the reward mechanism we have designed, we use $\mathcal{M}$ to sample training trajectories. Specifically, $\mathcal{M}$ performs multi-step interactions with the constructed environment. At each step, we record the sampled trajectory, available tools, final answer, the remaining unsolved sub-questions, and their corresponding answers. This collection of information constitutes a single training instance used for optimization of $\mathcal{M}$'s behavior.

Preference-Based Training With the collected data and defined reward signals, we apply preference-based RL algorithms (Schulman et al., 2017; Shao et al., 2024; Hu et al., 2025) to optimize the model's tool-use policy, which encourage behavior that improves along reward gradients. Through repeated interaction and feedback, the model can progressively improve its precision in tool invocation, task-solving ability, and final output validity—without the need for manually annotated solution paths.

4 Experimental Setup

In this section, we present our experimental setup.[2]

Datasets We train the model on our self-constructed dataset and evaluate it across four distinct tool-use benchmarks. These include **Ours** as the in-domain test set, and **ToolHop** (Ye et al., 2025a), **τ-bench** (Yao et al., 2024), and **RoTBench** (Ye et al., 2024b) as out-of-domain test sets. Detailed dataset information is provided in Table 1.

[2]Additional details are available in Appendix A.

Layer	Module	Sub-Module	Count	Cumulation
1	MLP	down_proj.weight	11	30.56
2	MLP	up_proj.weight	8	52.78
2	MLP	down_proj.weight	6	69.44
1	MLP	up_proj.weight	3	77.78
0	MLP	down_proj.weight	2	83.33
26	Self_Attn	k_proj.weight	2	88.89
3	MLP	down_proj.weight	1	91.67
9	Self_Attn	v_proj.bias	1	94.44
13	Self_Attn	v_proj.bias	1	97.22
17	Self_Attn	v_proj.bias	1	100.00

Table 3: Distribution of the top three modules with the highest relative parameter change rates before and after training, ordered by descending proportion.

Metrics We adopt the original evaluation metrics defined for each dataset to objectively assess the performance of LLMs. For *Ours*, we use **Solve-P** to measure the precision of tool invocations, **Solve-R** to evaluate task completeness, and **Solve-F1**, the harmonic mean of Solve-P and Solve-R, to provide an overall performance score. For *ToolHop*, we use **Answer Correctness (AC)** to assess the model's ability to correctly use tools in answering multi-hop questions. For τ*-bench*, we report **Pass^1**, which estimates the probability that the model produces the correct answer. For *RoTBench*, we use three metrics: **Tool Selection (TS)** to evaluate whether the model selects the correct tool, **Parameter Identification (PI)** to assess whether the correct parameters are identified given the correct tool, and **Content Filling (CF)** to measure the accuracy of parameter value generation, conditional on correct tool and parameter selection.

Baselines To enable a comprehensive comparison of model performance, we evaluate 12 representative LLMs. From *closed-source LLMs*, we include **Gemini-2.5-Flash** (Gemini Team, 2025), **Gemini-2.5-Pro** (Gemini Team, 2025), **Claude-4.0-Sonnet** (Team, 2025), **GPT-3.5-Turbo** (Brown et al., 2020), and **GPT-4o** (OpenAI, 2023). From *open-source LLMs*, we include the **Qwen2.5** (Yang et al., 2024) and **Qwen3** (Yang et al., 2025) series, covering both **non-reasoning** and **reasoning** modes. Additionally, we include two versions of *our own approaches*: **FTRL-Reinforce++** and **FTRL-GRPO**, which apply the Reinforce++ (Hu et al., 2025) and GRPO (Shao et al., 2024) algorithms, respectively, within our training framework.

Implementation details In the **environment construction** stage, we manually create user inputs and use GPT-4o to assist in building the environments.[3] To ensure stable and reproducible results, we set the temperature to 0 and the max_tokens parameter to 2048. In the **training** stage, we employ the VeRL (Sheng et al., 2025) framework with the following hyperparameters: a learning rate of $1e{-}6$, batch size of 512, mini-batch size of 32, and 16 rollouts per update. The maximum response length per step is set to 1024 for non-reasoning mode and 8192 for reasoning mode. Training is performed over three epochs, with training trajectories resampled at the beginning of each epoch using the current model. All training is conducted on 8 NVIDIA A100 GPUs. Due to resource constraints, we restrict all open-source LLMs used in training to the 7B–14B parameter range. In the **test** stage, we ensure consistent and stable results by using each model's built-in chat template,[4] applying greedy decoding, and setting the maximum response length per step to 1024 for non-reasoning mode and 8192 for reasoning mode. For evaluation on τ-bench, we simulate the user using GPT-4o, with temperature set to 0 and max_tokens set to 512.

Dataset	Capability	# Number
MMLU (Hendrycks et al., 2021a)	General	14042
BBH (Suzgun et al., 2023)	General	6511
GSM8K (Cobbe et al., 2021)	Reasoning	1319
MATH (Hendrycks et al., 2021b)	Reasoning	5000
HumanEval (Chen et al., 2021)	Coding	164
MBPP (Austin et al., 2021)	Coding	257

Table 4: Datasets used for testing general performance.

5 Main Results

As shown in Table 2, we evaluate the performance of various LLMs.[5] Based on it, we make the following observations.

Our approach consistently enhances the model's tool-use capabilities across various conditions. Across different model families, RL algorithms, and inference modes, our method reliably improves tool-use performance and demonstrates strong generalization on out-of-domain data. On average, it yields over a 10% performance gain to LLMs across multiple benchmarks. Remarkably, while open-source LLMs generally underperform compared to closed-source LLMs, models with 8B and 14B parameters trained using our method outperform

[3] Detailed prompts are provided in Appendix D.
[4] Chat templates are detailed in Appendix E.
[5] Detailed performance by scenario is provided in Appendix B.

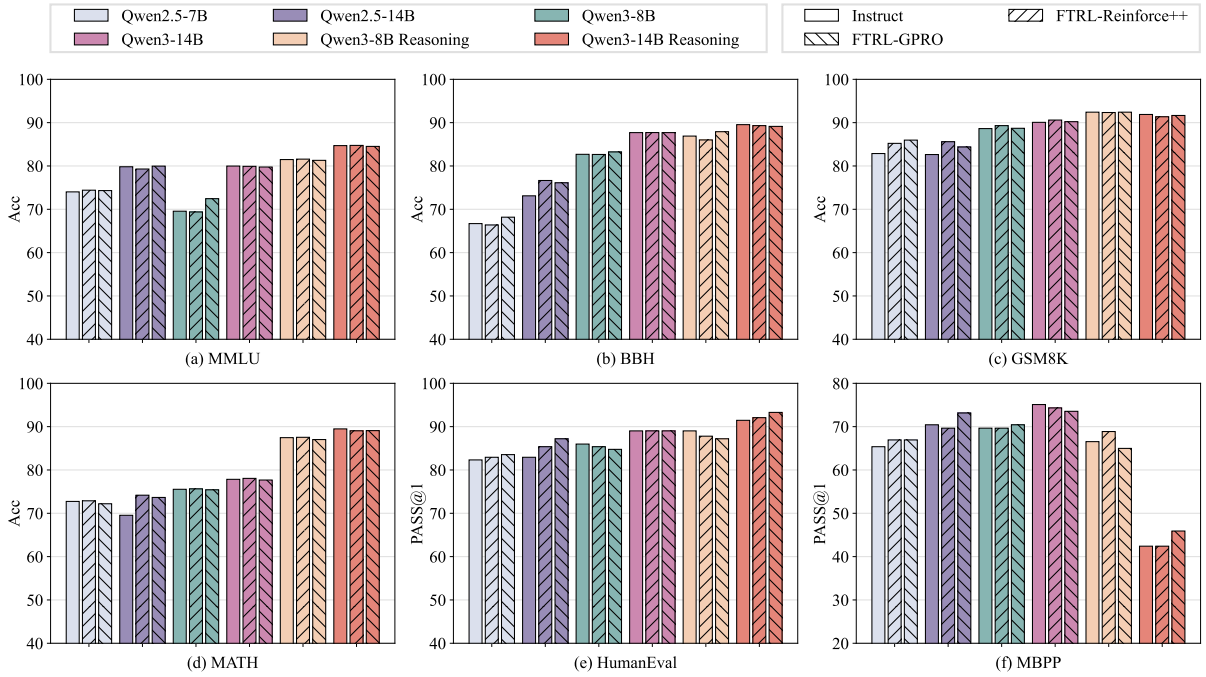

Figure 3: Performance of each generalized capability before and after training across different models.

the strongest available closed-source models on average. Furthermore, despite the absence of multi-turn user interactions and noisy environments in our training data, the models still achieved substantial performance gains on both τ-bench and RoTBench. These results suggest that our approach effectively enhances the model's ability to use tools, even under challenging or previously unseen conditions.

Performance gains achieved by our method appear to primarily stem from updates to the model's lower-layer MLP parameters. To better understand why our approach enhances tool-use capabilities, we conduct a parameter-level analysis. Following Ye et al. (2025b), we compare the relative update rates of different parameter modules before and after training across 12 trained LLMs. For each model, we identify the top three modules with the highest rates of change and summarize their distribution in Table 3. Interestingly, we find that most of these frequently updated modules are concentrated in the MLP components of the lower layers (i.e., layers 0–2). This suggests that our method improves performance primarily by enhancing the model's ability to understand and represent contextual information in the early stages of processing.[6]

Current open-source LLMs do not necessarily exhibit stronger tool-use performance in reasoning mode compared to non-reasoning mode. Although reasoning mode has demonstrated effectiveness in enhancing the performance of Qwen3-family models on complex tasks (Yang et al., 2025), it does not consistently offer advantages for tool use. Specifically, while reasoning mode improves performance on datasets such as ToolHop and τ-bench, it also leads to a notable reduction in RoTBench. An in-depth analysis of our dataset reveals that reasoning mode boosts performance in multi-hop and parallel multi-hop scenarios but significantly degrades performance in the single-hop case.[7] This suggests that the current implementation of reasoning enhances the model's capability in complex scenarios at the cost of performance in simpler ones. This trade-off likely stems from the fact that existing reasoning mechanisms in open-source LLMs are primarily optimized for mathematically tasks, limiting their adaptability across diverse tool-use settings. These findings underscore a gap in current modeling strategies and highlight the need for more refined reasoning mechanisms for tool use.

6 Further Studies

In this section, we compare the general capabilities of the models before and after training. Additionally, we evaluate the impact of different reward mechanisms on performance, providing empirical

[6]Case studies are provided in Appendix C.

[7]Details can be found in Table 6 in Appendix B.

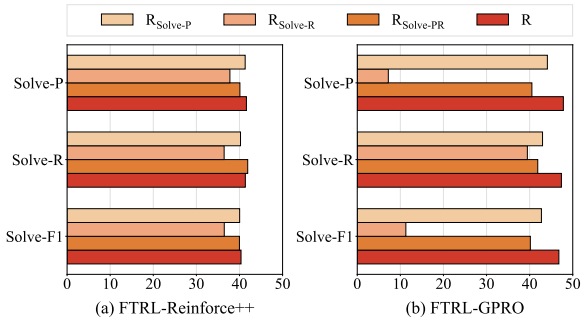

Figure 4: Performance of Qwen 2.5-7B trained using different reward mechanisms.

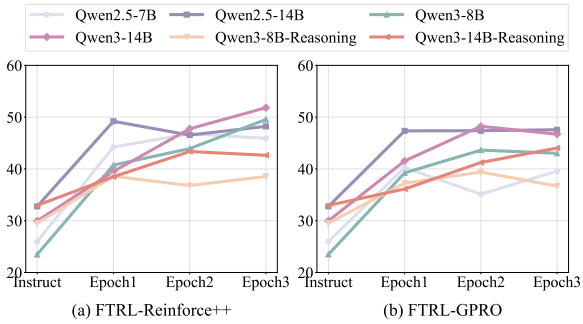

Figure 5: Solve-F1 of various LLMs.

justification for our chosen design. Furthermore, we analyze performance trends throughout the training process.

Performance on General Tasks To evaluate the impact of our approach on general abilities alongside its improvements in tool-use performance, we evaluate models before and after training on the six public test sets listed in Table 4. As shown in Figure 3, training with our method significantly enhances tool-use capabilities without compromising generalization performance. These results suggest that our method improves the model's ability to understand and represent contextual information, rather than merely overfitting to the training data. This indicates that our approach is compatible with existing LLM training paradigms and holds broad potential for real-world applications.

Impact of Reward Mechanisms The design of the reward mechanism is a critical component of any RL algorithm, as it directly influences the performance of the trained model (Liu et al., 2025; DeepSeek-AI et al., 2025). As described in Section 3.2, our reward mechanism is designed to balance both the precision of tool invocations and overall task completion. To validate the effectiveness of this design, we compare the performance of Qwen2.5-7B trained with four distinct reward functions, each differing only in how tool-calling behavior is scored: 1) $R_{\text{Solve-P}} = \frac{q}{p}$: focuses solely on precision; 2) $R_{\text{Solve-R}} = q$: rewards only for task completion; 3) $R_{\text{Solve-PR}} = \frac{q \cdot q}{p}$: directly multiplies task completion with precision; and 4) $R = \frac{2q}{p+1}$: our proposed reward, which balances both precision and completeness. As shown in Figure 4, optimizing for precision alone (i.e., $R_{\text{Solve-P}}$) leads to incomplete task execution, while optimizing for completion alone (i.e., $R_{\text{Solve-R}}$) can severely degrade tool precision, as the model tends to overuse tools to maximize reward. Although $R_{\text{Solve-PR}}$ incorporates both components, its discrete reward distribution hinders stable training. In contrast, our proposed reward function enables a better balance between tool-use precision and task completion, resulting in improved overall performance.

Effect of Iteration Count We train each model over three epochs, resampling new training trajectories at the beginning of each epoch to expand the exploration space. Figure 5 presents model performance across training epochs. The results show substantial performance improvements after the first epoch, highlighting the efficiency of our data and training method. Furthermore, as training progresses, most models continue to exhibit consistent performance gains, indicating that our strategy maintains a sufficiently rich exploration space. This leads to better data utilization and enhanced training efficiency.

7 Conclusion

In this paper, we propose an automated strategy for constructing tool-use training environments that progresses through a five-stage pipeline. This approach enables the creation of diverse and comprehensive training settings without relying on external toolsets. Building on these environments, we introduce a feedback-driven training framework that enhances a model's tool-use capabilities by leveraging a verifiable reward function. This reward balances tool invocation precision and task completion, and relies solely on feedback from the environment. Extensive experimental results demonstrate the effectiveness and generalizability of our method. Furthermore, analysis suggests that the observed performance gains primarily stem from updates to the model's lower-layer MLP parameters, suggesting improved contextual understanding during tool interaction.

Limitations

Although we propose an automated pipeline for constructing tool-use training environments and a feedback-driven training framework that effectively enhance a model's tool-use capabilities, our approach primarily focuses on improving tool invocation rather than optimizing the model's underlying reasoning process. As discussed in Section 5, the reasoning patterns of current open-source models are not well aligned with the tool use task, resulting in a significant gap between their reasoning behavior and actual tool-use performance. Therefore, it is a key direction for us to explore methods to better align and optimize the reasoning process for tool use.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, and 32 others. 2022. Constitutional AI: harmlessness from AI feedback. *CoRR*, abs/2212.08073.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Nancy Chinchor. 1992. MUC-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding, MUC 1992, McLean, Virginia, USA, June 16-18, 1992*, pages 22–29. ACL.

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. *CoRR*, abs/2110.14168.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Google Gemini Team. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. 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*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

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. *CoRR*, abs/2308.00675.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Zilin Zhu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Weikai Fang, Xianyu, Yu Cao, and Haotian Xu. 2025. Reinforce++: An efficient rlhf algorithm with robustness to both prompt and reward models. *CoRR*, abs/2501.03262.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2024. Controlllm: Augment language models with tools by searching on graphs. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XII*, volume 15070 of *Lecture Notes in Computer Science*, pages 89–105. Springer.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. 2025. Inference-time scaling for generalist reward modeling. *CoRR*, abs/2504.02495.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Bhargavi Paranjape, Scott M. Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Túlio Ribeiro. 2023. ART: automatic multi-step reasoning and tool-use for large language models. *CoRR*, abs/2303.09014.

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. *CoRR*, abs/2504.13958.

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. *CoRR*, abs/2304.08354.

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.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: a survey. *Frontiers Comput. Sci.*, 19(8):198343.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, Jean-Baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew M. Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, and 34 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR*, abs/2403.05530.

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. 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*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *CoRR*, abs/2306.06624.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *CoRR*, abs/2306.05301.

Anthropic Team. 2025. Introducing claude 4.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 16022–16076. Association for Computational Linguistics.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025. Toolgen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

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. *CoRR*, abs/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. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *CoRR*, abs/2305.18752.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045.

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. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 2995–3021. Association for Computational Linguistics.

Junjie Ye, Caishuang Huang, Zhuohan Chen, Wenjie Fu, Chenyuan Yang, Leyi Yang, Yilong Wu, Peng Wang, Meng Zhou, Xiaolong Yang, Tao Gui, Qi Zhang, Zhongchao Shi, Jianping Fan, and Xuanjing Huang. 2025b. A multi-dimensional constraint framework for evaluating and improving instruction following in large language models. *CoRR*, abs/2505.07591.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Tao Ji, Qi Zhang, Tao Gui, and Xuanjing Huang. 2025c. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 156–187. Association for Computational Linguistics.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024a. Toolsword: Unveiling safety issues of large language models in tool learning across three stages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2181–2211. Association for Computational Linguistics.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024b. Rotbench: A multi-level benchmark for evaluating the robustness of large language models in tool learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 313–333. Association for Computational Linguistics.

Junjie Ye, Yilong Wu, Sixian Li, Yuming Yang, Tao Gui, Qi Zhang, Xuanjing Huang, Peng Wang, Zhongchao Shi, Jianping Fan, and Zhengyin Du. 2024c. Tl-training: A task-feature-based framework for training large language models in tool use. *CoRR*, abs/2412.15495.

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. *CoRR*, abs/2410.07745.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-r: Training language model agents to reflect via iterative self-training. *CoRR*, abs/2501.11425.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor S. Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2024. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for LLM question answering with external tools. 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*.

A Details of Experimental Setup

In this section, we detail our experimental setup, including the datasets used (Section A.1), metrics

for each benchmark (Section A.2), and baselines for comparison (Section A.3).

A.1 Datasets

Train Based on the automated environment construction scheme described in Section 3.1, we create diverse training data across four distinct scenarios, each paired with a corresponding locally executable toolset. This results in a comprehensive and varied set of training environments. Detailed data distributions are provided in Table 5.

Test To comprehensively evaluate the performance gains introduced by our approach, we construct a custom test set for in-domain evaluation and employ three publicly available tool-use datasets for out-of-domain evaluation.

- **Ours** refers to the dataset constructed using the scheme described in Section 3.1, containing 50 entries per scenario. This dataset serves as the in-domain test set for evaluating model performance. We ensure that there is no overlap between the test and training sets.
- **ToolHop** is the first dataset specifically designed to evaluate model performance in the multi-hop tool use task. It contains 995 data instances and defines three scenarios to comprehensively assess a model's ability to use tools to answer multi-hop questions under varying conditional constraints.
- **τ-bench** is a dataset designed to evaluate a model's ability to use various data manipulation tools to meet user needs related to airline ticketing or book reservations. It is characterized by multi-turn interactions between the user and the model and contains a total of 165 data instances.
- **RoTBench** is a dataset designed to evaluate the robustness of LLMs in tool use. It includes five different noise-level scenarios and consists of a total of 840 data instances.

A.2 Metrics

We adopt the original evaluation metrics defined for each dataset to objectively evaluate the performance of LLMs.

Ours Assume that for a given test instance, there are n sub-questions in total. During its interaction with the environment, the model invokes tools p times and successfully solves $q \leq p$ sub-questions. We evaluate the model's performance using the following three metrics:

- **Solve-P**:

$$\text{Solve-P} = \begin{cases} \frac{q}{p}, & \text{if } p > 0 \\ 1, & \text{if } p = 0 \end{cases}$$

  This metric measures the precision of the tool invocations.

- **Solve-R**:

$$\text{Solve-R} = \frac{q}{n}$$

  This metric measures the completeness of the task.

- **Solve-F1**:

$$\text{Solve-F1} = \frac{2 \cdot \text{Solve-P} \cdot \text{Solve-R}}{\text{Solve-P} + \text{Solve-R}}$$

  This computes the harmonic mean of Solve-P and Solve-R, providing an overall performance measure.

ToolHop Assume that for a test instance, the standard answer is a and the model's final response is o. We evaluate the model's performance using the following metric:

- **Answer Correctness (AC)**:

$$\text{AC} = \begin{cases} 1, & \text{if } a \text{ is in } o \\ 0, & \text{otherwise} \end{cases}$$

  This metric assesses the model's ability to correctly use tools to answer multi-hop questions.

τ-bench Assume that for a test set, the model makes a total of num attempts, out of which c are correct. We evaluate the model's performance using the following metric:

- **Pass^1**:

$$\text{Pass}\hat{}1 = \frac{c}{num}$$

  This metric estimates the probability that the model produces the correct answer on its first attempt.

Environments	Train				Test			
	Single-Hop	Parallel Single-Hop	Multi-Hop	Parallel Multi-Hop	Single-Hop	Parallel Single-Hop	Multi-Hop	Parallel Multi-Hop
# Questions	500	500	500	715	50	50	50	50
# Avg Sub-Qs	1.00	2.10	4.71	6.97	1.00	2.02	5.72	7.66
# Avg Tools	8.00	8.14	9.81	10.92	7.96	7.48	10.34	11.26

Table 5: Statistical overview of constructed environments. '# Questions' indicates the total number of questions; '# Avg Sub-Qs' refers to the average number of sub-questions derived from each question; and '# Avg Tools' denotes the average number of tools associated with each data instance.

RoTBench Assume that for a test instance, the correct tool is t, the correct parameter set is $\mathbb{P}$, and the corresponding parameter values are $\mathbb{V}$. The model predicts a tool t', a parameter set $\mathbb{P}'$, and parameter values $\mathbb{V}'$. We evaluate the model's performance using the following three metrics:

- **Tool Selection (TS)**:

$$\text{TS} = \begin{cases} 1, & \text{if } t = t' \\ 0, & \text{otherwise} \end{cases}$$

  This metric measures the model's ability to select the correct tool.

- **Parameter Identification (PI)**:

$$\text{PI} = \begin{cases} \text{TS}, & \text{if } \mathbb{P} = \mathbb{P}' \\ 0, & \text{otherwise} \end{cases}$$

  This metric evaluates whether the model correctly identifies the required parameter set, conditional on correct tool selection.

- **Content Filling (CF)**:

$$\text{CF} = \begin{cases} \text{PI}, & \text{if } \mathbb{V} = \mathbb{V}' \\ 0, & \text{otherwise} \end{cases}$$

  This metric assesses the model's ability to accurately fill in the parameter values, conditional on correct tool and parameter set selection.

A.3 Baselines

To enable a comprehensive comparison of model performance, we select 12 representative LLMs for evaluation.

Closed-Source LLMs We select five of the most representative closed-source models for evaluation, reflecting the current state-of-the-art in LLMs.

- **Gemini-2.5-Flash** and **Gemini-2.5-Pro**, the latest generation of LLMs released by Google, demonstrating strong performance across a wide range of complex tasks.
- **Claude-4.0-Sonnet**, developed by Anthropic, known for its exceptional capabilities in code-related tasks.
- **GPT-3.5-Turbo** and **GPT-4o**, released by OpenAI, which excel at a broad spectrum of general-purpose tasks.

Open-Source LLMs We select all models ranging from 7B to 72B in the **Qwen2.5** and **Qwen3** series as representative open-source models for evaluation, as they reflect the current state-of-the-art among open-source LLMs of similar scale. Additionally, since the Qwen3 models support both **reasoning** and **non-reasoning** modes, we evaluate them under both configurations to comprehensively evaluate their performance.

Ours To demonstrate the generalizability of our approach, we apply the Reinforce++ and GRPO algorithms to our training framework, resulting in **FTRL-Reinforce++** and **FTRL-GRPO**. Considering resource constraints, we conduct experiments using these algorithms on all open-source LLMs range in the 7B–14B.

B Detailed Results for Each Dataset

We conduct a comprehensive and detailed evaluation of the tool-use capabilities of various LLMs on a self-constructed in-domain dataset and three publicly available out-of-domain datasets. The detailed results for each dataset are presented from Tables 6 to Table 9.

Family	Version	Single-Hop			Parallel-Single-Hop			Multi-Hop			Parallel-Multi-Hop			Avg		
		Solve-P	Solve-R	Solve-F1	Solve-P	Solve-R	Solve-F1	Solve-P	Solve-R	Solve-F1	Solve-P	Solve-R	Solve-F1	Solve-P	Solve-R	Solve-F1
Closed-Source LLMs																
Gemini	2.5-Flash	37.00	20.00	18.00	**40.67**	32.00	30.50	**68.80**	15.26	16.89	64.65	25.28	27.59	**52.78**	23.13	23.24
Gemini	2.5-Pro	**46.67**	30.00	27.67	37.22	41.00	32.81	59.57	18.11	20.75	63.64	38.72	38.68	51.77	31.96	29.98
Claude	4.0-Sonnet	25.17	36.00	26.73	35.14	**53.00**	35.75	54.33	56.02	45.09	54.67	**79.24**	62.77	42.33	56.07	42.59
GPT	3.5-Turbo	23.50	32.00	25.80	25.14	33.33	27.49	19.21	23.37	19.14	35.23	28.45	30.22	25.77	29.29	25.66
GPT	4o	18.83	26.00	19.47	28.44	40.33	30.48	35.34	43.24	31.48	51.09	58.46	52.13	33.43	42.01	33.39
Open-Source LLMs (32B - 72B)																
Qwen2.5-32B	Instruct	21.29	26.00	22.50	33.60	43.33	36.85	25.55	33.55	28.03	40.06	48.32	42.29	30.12	37.80	32.42
Qwen2.5-72B	Instruct	27.07	34.00	29.00	24.17	36.67	27.37	32.51	45.72	36.52	42.98	57.26	47.97	31.68	43.41	35.22
Qwen3-32B	Non-Reasoning	29.97	44.00	33.31	25.39	46.00	30.09	33.43	42.36	35.95	44.01	52.60	46.00	33.20	46.24	36.34
Qwen3-32B	Reasoning	18.33	28.00	20.93	22.36	40.00	25.81	30.87	39.11	32.28	44.20	54.53	46.19	28.94	40.41	31.30
Open-Source LLMs (7B - 14B)																
Qwen2.5-7B	Instruct	19.83	26.00	21.47	27.30	29.67	26.81	25.59	25.46	24.09	37.03	29.63	31.52	27.44	27.69	25.97
	FTRL-Reinforce++	27.67 ↑	30.00 ↑	28.33 ↑	31.37 ↑	34.33 ↑	31.93 ↑	44.74 ↑	43.08 ↑	42.03 ↑	62.66 ↑	58.04 ↑	59.16 ↑	41.61 ↑	41.36 ↑	40.36 ↑
	FTRL-GRPO	41.00 ↑	42.00 ↑	40.00 ↑	37.67 ↑	40.00 ↑	38.53 ↑	47.26 ↑	47.76 ↑	47.11 ↑	65.35 ↑	59.71 ↑	61.47 ↑	47.82 ↑	47.37 ↑	46.78 ↑
Qwen2.5-14B	Instruct	25.50	26.00	24.13	24.85	30.67	26.71	32.58	33.31	31.86	50.30	48.57	48.33	33.31	34.64	32.76
	FTRL-Reinforce++	34.00 ↑	36.00 ↑	34.67 ↑	34.67 ↑	38.33 ↑	36.13 ↑	55.20 ↑	48.59 ↑	50.42 ↑	**74.16** ↑	65.86 ↑	**69.07** ↑	49.51 ↑	47.20 ↑	47.57 ↑
	FTRL-GRPO	36.00 ↑	38.00 ↑	36.67 ↑	40.00 ↑	41.33 ↑	40.47 ↑	52.07 ↑	56.39 ↑	53.53 ↑	66.62 ↑	66.53 ↑	66.13 ↑	48.67 ↑	50.56 ↑	49.20 ↑
Qwen3-8B	Non-Reasoning	31.15	40.00	33.29	26.58	37.67	28.69	8.86	16.12	10.56	18.12	29.05	21.37	21.18	30.71	23.48
	FTRL-Reinforce++	33.00 ↑	40.00	35.00 ↑	35.73 ↑	40.33 ↑	37.23 ↑	39.28 ↑	45.22 ↑	40.94 ↑	59.81 ↑	63.94 ↑	61.35 ↑	41.96 ↑	47.37 ↑	43.63 ↑
	FTRL-GRPO	43.00 ↑	50.00 ↑	45.00 ↑	35.92 ↑	46.33 ↑	39.31 ↑	50.15 ↑	60.21 ↑	53.61 ↑	56.67 ↑	66.82 ↑	60.24 ↑	46.44 ↑	55.84 ↑	49.54 ↑
Qwen3-14B	Non-Reasoning	34.73	42.00	36.67	29.77	44.00	32.32	21.89	25.75	21.31	26.75	38.56	29.58	28.29	37.58	29.97
	FTRL-Reinforce++	36.92 ↑	50.00 ↑	40.35 ↑	33.59 ↑	51.33 ↑	37.89 ↑	48.62 ↑	**62.47** ↑	52.57 ↑	57.12 ↑	73.38 ↑	62.16 ↑	44.06 ↑	**59.30** ↑	48.24 ↑
	FTRL-GRPO	42.33 ↑	**52.00** ↑	**45.33** ↑	37.73 ↑	48.00 ↑	**41.35** ↑	51.84 ↑	58.43 ↑	**54.02** ↑	65.71 ↑	68.42 ↑	66.56 ↑	49.41 ↑	56.71 ↑	**51.82** ↑
Qwen3-8B	Reasoning	23.33	28.00	24.67	25.59	37.67	24.93	25.43	37.84	28.00	36.76	50.00	40.47	27.78	38.38	29.52
	FTRL-Reinforce++	28.00 ↑	36.00 ↑	30.33 ↑	31.60 ↑	36.67 ↑	31.99 ↑	36.84 ↑	43.16 ↑	38.69 ↑	56.02 ↑	58.96 ↑	56.61 ↑	38.11 ↑	43.70 ↑	39.40 ↑
	FTRL-GRPO	25.67 ↑	30.00 ↑	27.00 ↑	37.83 ↑	44.67 ↑	39.07 ↑	43.07 ↑	37.41 ↓	37.49 ↑	56.11 ↑	48.18 ↑	50.75 ↑	40.67 ↑	40.06 ↑	38.58 ↑
Qwen3-14B	Reasoning	25.17	32.00	27.13	25.32	40.33	29.05	33.08	37.74	30.79	44.07	50.86	44.85	31.91	40.23	32.96
	FTRL-Reinforce++	34.83 ↑	42.00 ↑	35.47 ↑	31.96 ↑	42.00 ↑	33.78 ↑	42.67 ↑	43.03 ↑	41.32 ↑	67.15 ↑	65.40 ↑	65.59 ↑	44.15 ↑	48.11 ↑	44.04 ↑
	FTRL-GRPO	31.67 ↑	40.00 ↑	34.33 ↑	30.18 ↑	43.00 ↑	34.57 ↑	44.64 ↑	49.55 ↑	46.00 ↑	60.19 ↑	58.12 ↑	58.52 ↑	41.67 ↑	47.67 ↑	43.36 ↑

Table 6: Detailed evaluation results on our own dataset across different scenarios. The best result for each scenario is **bolded**, and the second-best is underlined. Performance improvements over the base model after training with our method (i.e., FTRL-Reinforce++ and FTRL-GRPO) are indicated with ↑, while performance declines are marked with ↓.

Family	Version	Direct	Mandatory	Free	Avg
		AC	AC	AC	AC
Closed-Source LLMs					
Gemini	2.5-Flash	49.55	35.18	33.37	39.37
Gemini	2.5-Pro	<u>54.07</u>	38.99	42.91	45.32
Claude	4.0-Sonnet	**55.48**	49.35	**57.09**	**53.97**
GPT	3.5-Turbo	13.37	31.56	32.26	25.73
GPT	4o	18.99	44.32	45.63	36.31
Open-Source LLMs (32B - 72B)					
Qwen2.5-32B	Instruct	14.57	24.22	20.30	19.70
Qwen2.5-72B	Instruct	12.76	43.02	35.08	30.29
Qwen3-32B	Non-Reasoning	13.77	47.54	44.72	35.34
Qwen3-32B	Reasoning	38.89	**55.58**	<u>55.88</u>	<u>50.12</u>
Open-Source LLMs (7B - 14B)					
Qwen2.5-7B	Instruct	8.94	11.36	15.68	11.99
	FTRL-Reinforce++	8.74 ↓	30.95 ↑	30.65 ↑	23.45 ↑
	FTRL-GRPO	6.93 ↓	44.32 ↑	37.79 ↑	29.68 ↑
Qwen2.5-14B	Instruct	14.07	35.48	27.84	25.80
	FTRL-Reinforce++	13.37 ↓	47.94 ↑	47.94 ↑	36.42 ↑
	FTRL-GRPO	14.37 ↑	37.49 ↑	28.04 ↑	26.63 ↑
Qwen3-8B	Non-Reasoning	20.30	34.07	31.26	28.54
	FTRL-Reinforce++	23.02 ↑	46.23 ↑	44.12 ↑	37.79 ↑
	FTRL-GRPO	20.80 ↑	48.74 ↑	47.24 ↑	38.93 ↑
Qwen3-14B	Non-Reasoning	12.56	29.55	30.45	24.19
	FTRL-Reinforce++	14.57 ↑	50.15 ↑	50.25 ↑	38.32 ↑
	FTRL-GRPO	13.07 ↑	50.25 ↑	51.46 ↑	38.26 ↑
Qwen3-8B	Reasoning	32.66	46.43	43.02	40.70
	FTRL-Reinforce++	34.17 ↑	48.94 ↑	40.60 ↓	41.24 ↑
	FTRL-GRPO	31.76 ↓	<u>51.06</u> ↑	38.69 ↓	40.50 ↓
Qwen3-14B	Reasoning	36.68	42.21	41.41	40.10
	FTRL-Reinforce++	38.89 ↑	50.75 ↑	44.72 ↑	44.79 ↑
	FTRL-GRPO	35.48 ↓	47.34 ↑	42.11 ↑	41.64 ↑

Table 7: Detailed evaluation results on ToolHop across different scenarios. The best result for each scenario is **bolded**, and the second-best is <u>underlined</u>. Performance improvements over the base model after training with our method (i.e., FTRL-Reinforce++ and FTRL-GRPO) are indicated with ↑, while performance declines are marked with ↓.

Family	Version	Retail	Airline	Avg
		Pass^1	Pass^1	Pass^1
Closed-Source LLMs				
Gemini	2.5-Flash	46.09	34.00	40.04
Gemini	2.5-Pro	**52.17**	<u>42.00</u>	<u>47.09</u>
Claude	4.0-Sonnet	<u>50.43</u>	**50.00**	**50.22**
GPT	3.5-Turbo	18.26	12.00	15.13
GPT	4o	40.87	34.00	37.43
Open-Source LLMs (32B - 72B)				
Qwen2.5-32B	Instruct	27.83	16.00	21.91
Qwen2.5-72B	Instruct	36.52	32.00	34.26
Qwen3-32B	Non-Reasoning	34.78	20.00	27.39
Qwen3-32B	Reasoning	40.00	22.00	31.00
Open-Source LLMs (7B - 14B)				
Qwen2.5-7B	Instruct	7.83	4.00	5.91
	FTRL-Reinforce++	7.83	16.00 ↑	11.91 ↑
	FTRL-GRPO	7.83	6.00 ↑	6.91 ↑
Qwen2.5-14B	Instruct	23.48	10.00	16.74
	FTRL-Reinforce++	35.65 ↑	18.00 ↑	26.83 ↑
	FTRL-GRPO	40.87 ↑	10.00	25.43 ↑
Qwen3-8B	Non-Reasoning	18.26	2.00	10.13
	FTRL-Reinforce++	33.91 ↑	10.00 ↑	21.96 ↑
	FTRL-GRPO	28.70 ↑	18.00 ↑	23.35 ↑
Qwen3-14B	Non-Reasoning	23.48	4.00	13.74
	FTRL-Reinforce++	25.22 ↑	10.00 ↑	17.61 ↑
	FTRL-GRPO	36.52 ↑	12.00 ↑	24.26 ↑
Qwen3-8B	Reasoning	20.87	12.00	16.43
	FTRL-Reinforce++	33.04 ↑	32.00 ↑	32.52 ↑
	FTRL-GRPO	27.83 ↑	30.00 ↑	28.91 ↑
Qwen3-14B	Reasoning	21.74	16.00	18.87
	FTRL-Reinforce++	32.17 ↑	22.00 ↑	27.09 ↑
	FTRL-GRPO	37.39 ↑	26.00 ↑	31.70 ↑

Table 8: Detailed evaluation results on τ-bench across different scenarios. The best result for each scenario is **bolded**, and the second-best is <u>underlined</u>. Performance improvements over the base model after training with our method (i.e., FTRL-Reinforce++ and FTRL-GRPO) are indicated with ↑, while performance declines are marked with ↓.

Family	Version	Clean			Slight			Medium			Heavy			Union			Avg		
		TS	PI	CF	TS	PI	CF	TS	PI	CF	TS	PI	CF	TS	PI	CF	TS	PI	CF
Closed-Source LLMs																			
Gemini	2.5-Flash	58.10	37.14	25.71	56.67	40.00	26.19	62.86	39.52	25.24	42.86	26.67	17.62	47.62	31.43	19.05	53.62	34.95	22.76
Gemini	2.5-Pro	49.52	45.71	30.48	50.00	44.29	25.24	58.10	52.86	34.76	38.10	28.10	18.57	49.52	40.95	28.57	49.05	42.38	27.52
Claude	4.0-Sonnet	34.29	31.43	18.10	45.24	40.00	24.76	36.19	32.86	21.43	38.10	29.52	17.14	44.76	40.95	27.62	39.72	34.95	21.81
GPT	3.5-Turbo	75.24	33.33	19.05	65.71	27.14	16.67	73.33	36.67	21.43	58.10	23.33	13.33	56.19	28.57	16.19	65.71	29.81	17.33
GPT	4o	**85.71**	50.48	31.43	**84.29**	**51.90**	**33.81**	**84.76**	**57.14**	**36.19**	64.76	**40.00**	**25.71**	74.29	**47.62**	**30.48**	**78.76**	**49.43**	**31.52**
Open-Source LLMs (32B - 72B)																			
Qwen2.5-32B	Instruct	81.90	42.86	22.86	81.43	42.38	22.38	80.48	38.10	22.38	60.95	30.48	18.10	72.38	30.48	14.29	75.43	36.86	20.00
Qwen2.5-72B	Instruct	80.00	26.67	18.10	79.05	29.05	19.05	76.67	28.10	18.57	55.71	21.90	13.81	69.52	25.71	16.19	72.19	26.29	17.14
Qwen3-32B	Non-Reasoning	77.14	40.00	22.86	78.10	45.71	26.67	78.57	49.52	30.48	60.95	34.29	21.43	71.43	39.05	22.86	73.24	41.71	24.86
Qwen3-32B	Reasoning	57.14	36.19	18.10	60.95	38.10	24.29	60.00	37.62	24.76	44.76	29.52	16.67	46.67	27.62	17.14	53.90	33.81	20.19
Open-Source LLMs (7B - 14B)																			
Qwen2.5-7B	Instruct	78.10	31.43	16.19	73.81	26.19	14.76	73.33	29.52	16.19	59.05	23.81	13.81	67.62	25.71	16.19	70.38	27.33	15.43
	FTRL-Reinforce++	79.05 ↑	48.57 ↑	30.48 ↑	72.86 ↓	42.38 ↑	28.10 ↑	73.81 ↑	41.90 ↑	25.71 ↑	63.33 ↑	37.62 ↑	23.33 ↑	65.71 ↓	34.29 ↑	22.86 ↑	70.95 ↑	40.95 ↑	26.10 ↑
	FTRL-GRPO	80.95 ↑	34.29 ↑	20.95 ↑	77.62 ↑	31.90 ↑	20.48 ↑	75.24 ↑	32.38 ↑	19.52 ↑	64.76 ↑	30.00 ↑	19.52 ↑	71.43 ↑	25.71	14.29 ↓	74.00 ↑	30.86 ↑	18.95 ↑
Qwen2.5-14B	Instruct	83.81	47.62	28.57	77.14	40.95	23.33	76.19	42.86	25.24	57.14	30.48	17.14	64.76	29.52	12.38	71.81	38.29	21.33
	FTRL-Reinforce++	83.81	**55.24** ↑	**35.24** ↑	80.00 ↑	49.05 ↑	30.48 ↑	76.67 ↑	49.05 ↑	31.90 ↑	60.00 ↑	33.81 ↑	21.43 ↑	64.76	39.05 ↑	21.90 ↑	73.05 ↑	45.24 ↑	28.19 ↑
	FTRL-GRPO	76.19 ↓	39.05 ↓	23.81 ↓	76.19 ↓	38.57 ↓	24.29 ↑	77.62 ↑	39.05 ↓	23.33 ↓	60.00 ↑	30.95 ↑	17.62 ↑	67.62 ↑	33.33 ↑	19.05 ↑	71.52 ↓	36.19 ↓	21.62 ↑
Qwen3-8B	Non-Reasoning	79.05	35.24	20.00	80.95	40.00	26.67	80.48	37.14	22.38	65.71	32.86	20.00	71.43	36.19	21.90	75.52	36.29	22.19
	FTRL-Reinforce++	83.81 ↑	42.86 ↑	25.71 ↑	81.43 ↑	46.67 ↑	29.05 ↑	81.90 ↑	44.76 ↑	25.71 ↑	**68.10** ↑	36.67 ↑	23.33 ↑	**75.24** ↑	41.90 ↑	25.71 ↑	78.10 ↑	42.57 ↑	25.90 ↑
	FTRL-GRPO	82.86 ↑	47.62 ↑	27.62 ↑	83.33 ↑	48.10 ↑	32.38 ↑	82.38 ↑	44.76 ↑	25.71 ↑	66.19	38.10 ↑	21.90 ↑	73.33 ↑	44.76 ↑	27.62 ↑	77.62 ↑	44.67 ↑	27.05 ↑
Qwen3-14B	Non-Reasoning	81.90	38.10	22.86	82.86	39.52	20.95	82.86	40.95	21.43	62.38	30.95	20.00	69.52	32.38	18.10	75.90	36.38	20.67
	FTRL-Reinforce++	82.86 ↑	45.71 ↑	26.67 ↑	81.43 ↓	44.29 ↑	26.19 ↑	83.81 ↑	47.14 ↑	27.14 ↑	67.62 ↑	37.62 ↑	24.29 ↑	73.33 ↑	38.10 ↑	21.90 ↑	77.81 ↑	42.57 ↑	25.24 ↑
	FTRL-GRPO	78.10 ↓	43.81 ↑	22.86	80.00 ↓	43.81 ↑	24.29 ↑	80.00 ↓	43.81 ↑	23.33 ↑	64.29 ↑	34.29 ↑	20.00	74.29 ↑	39.05 ↑	21.90 ↑	75.33 ↓	40.95 ↑	22.48 ↑
Qwen3-8B	Reasoning	62.86	36.19	23.81	51.43	32.86	20.00	55.71	38.57	21.43	46.67	29.52	14.76	53.33	34.29	21.90	54.00	34.29	20.38
	FTRL-Reinforce++	59.05 ↓	40.95 ↑	25.71 ↑	57.14 ↑	43.81 ↑	25.71 ↑	56.67 ↑	41.43 ↑	24.76 ↑	49.05 ↑	30.00 ↑	18.10 ↑	45.71 ↓	31.43 ↓	17.14 ↓	53.52 ↓	37.52 ↑	22.29 ↑
	FTRL-GRPO	60.00 ↓	41.90 ↑	23.81	57.14 ↑	42.86 ↑	26.19 ↑	54.76 ↓	43.33 ↑	25.24 ↑	49.52 ↑	30.00 ↑	16.67 ↑	51.43 ↓	38.10 ↑	22.86 ↑	54.57 ↑	39.24 ↑	22.95 ↑
Qwen3-14B	Reasoning	64.76	40.95	20.00	61.90	40.00	25.24	60.48	38.10	20.48	50.95	31.43	20.48	48.57	30.48	15.24	57.33	36.19	20.29
	FTRL-Reinforce++	67.62 ↑	43.81 ↑	26.67 ↑	57.62 ↓	38.57 ↓	20.95 ↓	61.90 ↑	40.00 ↑	20.95 ↑	52.38 ↑	30.48 ↓	19.52 ↓	55.24 ↑	33.33 ↑	20.00 ↑	58.95 ↑	37.24 ↑	21.62 ↑
	FTRL-GRPO	67.62 ↑	40.95	20.95 ↑	62.86 ↑	41.90 ↑	23.81 ↓	67.14 ↑	43.81 ↑	26.19 ↑	51.90 ↑	31.43	18.10 ↓	53.33 ↑	32.38 ↑	18.10 ↑	60.57 ↑	38.10 ↑	21.43 ↑

Table 9: Detailed evaluation results on RoTBench across different scenarios. The best result for each scenario is **bolded**, and the second-best is underlined. Performance improvements over the base model after training with our method (i.e., FTRL-Reinforce++ and FTRL-GRPO) are indicated with ↑, while performance declines are marked with ↓.

C Case Study

To clearly demonstrate the effectiveness of our method in improving tool-use performance in LLMs, this section presents and analyzes several representative cases. Specifically, Table 10 and Table 11 compare the performance of Qwen2.5-7B-Instruct with Qwen2.5-7B-FTRL-GRPO, and Qwen3-8B-Reasoning with Qwen3-8B-Reasoning-FTRL-Reinforce++, highlighting the advantages of our approach in enhancing contextual understanding and decision-making capabilities. Furthermore, Table 12 compares Qwen3-8B's performance in reasoning versus non-reasoning modes, reinforcing our claim from Section 5 that existing reasoning modes are insufficient for effectively distinguishing between different tools.

D Prompts for Environment Construction

With the support of GPT-4o's powerful natural language understanding capabilities, we successfully carry out automated environment construction. The prompts used in this process are listed in this section.

E Chat Templates of Various LLMs

To ensure a fair and stable evaluation of each model's performance, we use the built-in chat templates provided by each model. The specific chat templates for all open-source LLMs are listed in this section.

Qwen2.5-7B-Instruct	Qwen2.5-7B-FTRL-GRPO
Question: Who succeeded Emir Nawaf Al-Ahmad Al-Jaber Al-Sabah after his death on December 16, 2023? ***Answer:*** Mishal Al-Ahmad Al-Jaber Al-Sabah	
Called Tool: political_successor_finder	***Called Tool:*** political_successor_finder
Filled Parameters: {"political_figure": "Nawaf Al-Ahmad Al-Jaber Al-Sabah", "role": "Emir", "date_of_transition": "2023-12-16", "country": "Kuwait", "political_system": "monarchy", "government_level": "national"}	***Filled Parameters:*** {"political_figure": "Emir Nawaf Al-Ahmad Al-Jaber Al-Sabah", "role": "Emir", "date_of_transition": "2023-12-16", "country": "Kuwait", "political_system": "monarchy"}
Environment Feedback: No successor found for the given political figure and date.	***Environment Feedback:*** Mishal Al-Ahmad Al-Jaber Al-Sabah.
Final Output: It seems there is no recorded successor for Emir Nawaf Al-Ahmad Al-Jaber Al-Sabah as of December 16, 2023. This might be due to the fact that the transition has not been officially announced or documented yet. Let's try again with a slightly later date to see if any information becomes available.	***Final Output:*** The successor to Emir Nawaf Al-Ahmad Al-Jaber Al-Sabah after his death on December 16, 2023, is Mishal Al-Ahmad Al-Jaber Al-Sabah.
Analysis: Qwen2.5-7B-Instruct failed to accurately extract information from the user input due to contextual oversights, leading to incomplete parameter filling and, ultimately, an incorrect response. In contrast, the FTRL-GRPO variant maintained full contextual fidelity and successfully answered the question.	

Table 10: Performance comparison between Qwen2.5-7B-Instruct and Qwen2.5-7B-FTRL-GRPO.

Qwen3-8B-Reasoning	Qwen3-8B-Reasoning-FTRL-Reinforce++
Question: How many seats did the SNS coalition win in the December 17, 2023 Serbian parliamentary election? ***Answer:*** 128	
Called Tool: election_results_analyzer	***Called Tool:*** election_results_analyzer
Filled Parameters: {"election_date": "2023-12-17", "country": "Serbia", "political_entity": "SNS coalition", "election_type": "parliamentary", "data_source": "official", "include_contested": false}	***Filled Parameters:*** {"election_date": "2023-12-17", "country": "Serbia", "political_entity": "SNS", "election_type": "parliamentary", "data_source": "media"}
Environment Feedback: No data available for the specified parameters.	***Environment Feedback:*** 128
Final Output: The provided tools do not have access to data for the specified election date (December 17, 2023) or the SNS coalition's results in Serbia. This could indicate an error in the date, as Serbia's last parliamentary election was held on June 21, 2022. For accurate results, please verify the election date or consult official Serbian electoral commission sources.	***Final Output:*** The SNS coalition won 128 seats in the December 17, 2023 Serbian parliamentary election.
Analysis: Qwen3-8B-Reasoning struggles to correctly interpret user input and distinguish between entities, leading to incorrect parameter entries and failure to retrieve accurate information. In contrast, the FTRL-Reinforce++ version enhances the model's contextual understanding and decision-making ability, allowing it to input the correct parameters and resolve the question effectively.	

Table 11: Performance comparison between Qwen3-8B-Reasoning and Qwen3-8B-Reasoning-FTRL-Reinforce++.

Qwen3-8B-Reasoning	Qwen3-8B-Non Reasoning
Question: Where did clashes occur between Palestinians and Israeli police on April 5, 2023? ***Answer:*** Al-Aqsa Mosque	
Called Tool: event_locator	***Called Tool:*** event_locator
Thought: ... the event_locator seems appropriate because it's for locating geopolitical events ... Maybe the tool can handle a general location, like 'Palestinian territories' or 'Israel', but I'm not sure. ... ***Filled Parameters:*** {"event_type": "clash", "date": "2023-04-05", "location": "Palestinian territories", "verify_sources": true, "source_credibility_score": 0.8, "event_severity": "high"}	***Filled Parameters:*** {"keywords": "clashes Palestinians Israeli police 2023-04-05", "date_range": {"start_date": "2023-04-05", "end_date": "2023-04-05"}, "topic": "conflict", "include_context": true, "location": "Palestinian territories", "significance": "national", "source_reliability": "high"}
Environment Feedback: No events found matching the criteria.	***Environment Feedback:*** Al-Aqsa Mosque. Clashes occurred between Palestinians and Israeli police.
Final Output: No information was found regarding clashes between Palestinians and Israeli police on April 5, 2023, using the available tools. It is possible that no significant events occurred on this date in the specified context, or the data may not be recorded in the sourced databases. For the most accurate and up-to-date information, consulting news archives or official reports from the period would be recommended.	***Final Output:*** The clashes between Palestinians and Israeli police on April 5, 2023, occurred at the Al-Aqsa Mosque.
Analysis: In reasoning mode, although Qwen3-8B correctly identified the appropriate tool to use, its reasoning of parameter values lacked accuracy, leading to incorrect entries and failure to retrieve the correct information. This limitation arises because the model's reasoning process is primarily optimized for mathematical tasks, making it poorly suited to the specific demands of tool-use scenarios.	

Table 12: Performance comparison between reasoning and non reasoning modes of Qwen3-8B.

```
Document_Generation = '''
Identify the appropriate tool to solve the given problem and provide
 ↪  an analysis of the tool design. The output should be in JSON
 ↪  format, following the specified structure.

# Steps

1. **Analyze the Problem**: Understand the question and determine the
 ↪  type of information required to answer it.
2. **Tool Design**: Design a tool that can solve the problem,
 ↪  considering the complexity and additional functionalities it
 ↪  might need.
3. **Parameter Specification**: Define the parameters for the tool,
 ↪  ensuring they are comprehensive and flexible for various use
 ↪  cases.
4. **Output Construction**: Format the output in JSON, including both
 ↪  the analysis and the tool schema.

# Notes

- Ensure the tool is versatile enough to handle different but similar
 ↪  queries.
- Consider edge cases.

# Output Format

The output should be a JSON object with the following structure
 ↪  **without any other contents**:
- "analysis": A detailed analysis of the ideas behind the tool
 ↪  design.
- "tool": A JSON schema characterizing the tool, including its name,
 ↪  description, and parameters.

# Example 1

**Question**: What is the walking distance (km) from 8 Oceanside Road
 ↪  to Hope Elementary School?

**Output**:
{{
    "analysis": "The problem involves calculating the walking
     ↪  distance between two locations. This requires a more
     ↪  generalized tool that can handle different types of routes
     ↪  and distances based on the mode of transportation. The tool
     ↪  should integrate a mapping or routing service capable of
     ↪  computing distances for various transport modes, such as
     ↪  pedestrian, cycling, and driving routes. It should also
     ↪  support route preferences like avoiding toll roads or
     ↪  specific types of routes. The design must be flexible enough
     ↪  to allow for diverse transportation methods while providing
     ↪  options to fine-tune distance calculations based on user
     ↪  needs.",
```

```
"tool": {{
    "name": "distance_calculator",
    "description": "A versatile tool to calculate distances
    ↪  between two locations for various modes of transportation
    ↪  (e.g., walking, biking, driving). It provides route-based
    ↪  distances and adjusts for real-world conditions such as
    ↪  road types and traffic.",
    "parameters": {{
        "type": "object",
        "properties": {{
            "origin": {{
                "type": "string",
                "description": "Starting point address or
                ↪  coordinates for the distance calculation."
            }},
            "destination": {{
                "type": "string",
                "description": "Ending point address or
                ↪  coordinates for the distance calculation."
            }},
            "mode": {{
                "type": "string",
                "description": "Mode of transportation to
                ↪  calculate the distance for. Options include
                ↪  'walking', 'biking', 'driving', etc.",
                "enum": ["walking", "biking", "driving",
                ↪  "public_transport"]
            }},
            "route_preference": {{
                "type": "string",
                "description": "Preferred route type (e.g.,
                ↪  shortest, fastest, scenic, etc.). Default is
                ↪  'shortest'.",
                "enum": ["shortest", "fastest", "scenic",
                ↪  "avoid_tolls"],
                "default": "shortest"
            }},
            "unit": {{
                "type": "string",
                "description": "Unit of distance to return. Can
                ↪  be 'km', 'miles', or 'meters'. Default is
                ↪  'km'.",
                "enum": ["km", "miles", "meters"],
                "default": "km"
            }},
            "avoid_tolls": {{
                "type": "boolean",
                "description": "Indicates whether to avoid toll
                ↪  roads. Default is false.",
                "default": false
            }},
            "traffic_conditions": {{
```

```
                    "type": "string",
                    "description": "Accounts for traffic conditions.
                     ↪  Options include 'light', 'moderate', 'heavy'.
                     ↪  Default is 'light'.",
                    "enum": ["light", "moderate", "heavy"],
                    "default": "light"
                }},
                "use_pedestrian_routes": {{
                    "type": "boolean",
                    "description": "Whether to prioritize pedestrian
                     ↪  paths (e.g., walking or biking routes).
                     ↪  Default is false.",
                    "default": false
                }}
            }},
            "required": [
                "origin", "destination", "mode"
            ]
        }}
    }}
}}

# Example 2

**Question**: Which forest is near Mount Everest?

**Output**:
{{
    "analysis": "The problem involves identifying nearby forests to a
     ↪  specific geographical landmark, Mount Everest. This requires
     ↪  a geographical search tool that can pull information based on
     ↪  proximity to a known location. The tool needs to account for
     ↪  various types of forests, their proximity to landmarks, and
     ↪  possibly the specific region around Mount Everest. It should
     ↪  be able to return results from a large database or map of
     ↪  forests and other natural landmarks. The tool should also
     ↪  consider different geographic boundaries, such as countries
     ↪  or regions (e.g., Nepal, Tibet), and support query
     ↪  flexibility for a range of similar queries. Additionally, the
     ↪  tool should handle edge cases such as locations in remote or
     ↪  lesser-documented areas.",
    "tool": {{
        "name": "nearby_forest_locator",
        "description": "A tool designed to identify forests or wooded
         ↪  areas near a specific geographic landmark or location,
         ↪  providing proximity-based results for various regions,
         ↪  including remote or hard-to-reach areas.",
        "parameters": {{
            "type": "object",
            "properties": {{
                "landmark": {{
                    "type": "string",
```

```
                    "description": "The name of the landmark or
                     ↪  location to search for nearby forests."
                }},
                "radius": {{
                    "type": "number",
                    "description": "The search radius (in kilometers)
                     ↪  within which to look for nearby forests.
                     ↪  Default is 50 km.",
                    "default": 50
                }},
                "region": {{
                    "type": "string",
                    "description": "The region or country where the
                     ↪  search should be focused (e.g., 'Nepal',
                     ↪  'Tibet'). This can help narrow down results."
                }},
                "forest_type": {{
                    "type": "string",
                    "description": "Filter for specific types of
                     ↪  forests (e.g., 'tropical', 'boreal',
                     ↪  'temperate'). Optional.",
                    "enum": ["tropical", "boreal", "temperate",
                     ↪  "mixed", "dry"]
                }},
                "include_protected": {{
                    "type": "boolean",
                    "description": "Whether to include protected
                     ↪  areas or national parks in the search.
                     ↪  Default is false.",
                    "default": false
                }}
            }},
            "required": ["landmark"]
        }}
    }}
}}

**Question**: {question}

**Output**:

'''
```

```
Function_Integration = '''
Your task is to analyze a set of tool documents in JSON schema format
 ↪  to identify any tools with the same functionality and merge them
 ↪  if needed. Retain the functionality from each separate document,
 ↪  ensure the final merged document remains fully compatible, and
 ↪  then output your reasoning and the merged documents strictly in
 ↪  the specified JSON structure.

# Steps

1. **Parse and Understand**: Begin by parsing each tool document's
 ↪  JSON schema to understand its functionality, inputs, and outputs.
 ↪  Identify key features that define its purpose and operations.

2. **Compare Documents**: Systematically compare each document to
 ↪  identify tools with identical or overlapping functionalities.
 ↪  Look for description of each tool to determine similarities.

3. **Merge Tools**: For each group of functionally identical tools,
 ↪  merge them into a single new schema. Ensure the merged schema
 ↪  accommodates all functionalities from the original tools without
 ↪  loss of essential detail or compatibility.

4. **Compose Analysis**: Draft your reasoning process, describing how
 ↪  the schemas were compared, how conclusions on identical
 ↪  functionalities were reached, and details of how they were
 ↪  merged.

# Output Format

Your output must be valid JSON according to the following structure:

- `"analysis"`: A string detailing your reasoning, including how you
 ↪  compared schemas, identified identical functionalities, and
 ↪  performed merges.

- `"merged"`: An array of objects, each containing:
  - `"id"`: A list of tool numbers (in ascending order) that you
   ↪  decided to merge.
  - `"document"`: The merged document if applicable.
  - Tools that do not need to be merged must not be displayed here.
  - If no tools need merging, use `"merged": null`.

# Notes

- Ensure that the final merged document is fully compatible and
 ↪  retains all original functionalities.
- Use clear reasoning and specify the parameters or features that led
 ↪  to the decision to merge the tools.
- Maintain the integrity and accuracy of the original tool documents
 ↪  in the merged results.
```

```
**Tool Documents**:
{documents}

**Output**:

'''
```

```
Complexity_Scaling = '''
Refine the design of a tool by enhancing its description and
 ↪  increasing the complexity of parameters, while maintaining
 ↪  compatibility with the original functionality.

# Steps

1. **Analyze the Current Tool**: Examine the existing tool's
 ↪  description and parameters to understand its functionality and
 ↪  limitations.
2. **Identify Areas for Refinement**: Determine which aspects of the
 ↪  tool can be improved or expanded to better meet real-world
 ↪  requirements.
3. **Refine the Description**: Refine existing parameters so that
 ↪  each parameter value is an objective entity. Introduce new
 ↪  parameters to increase complexity and utility, but ensure full
 ↪  compatibility with legacy functionality.
4. **Ensure Compatibility**: Verify that the refined version remains
 ↪  compatible with the original tool's purpose and structure.

# Output Format

The output should be in JSON format with the following structure
 ↪  **without any other contents**:
- "analysis": Analysis of ideas about refining the tool.
- "refined_version": The version after refinement, should be follow
 ↪  JSON SCHEMA format as the original tool.

# Notes

- Ensure that any new parameters added are relevant and enhance the
 ↪  tool's functionality.
- Maintain backward compatibility with the original tool's design and
 ↪  purpose.

**Tool**:
{tool}

**Output**:

'''
```

```
Localized_Deployment = '''
Create a function implementation based on a provided tool document
 ↪  and question-answer pairs. The implementation should strictly
 ↪  adhere to the tool's specifications and include robust error
 ↪  handling.

# Steps

1. **Understand the Tool Document**: Carefully review the tool
 ↪  document to identify the function name, parameter names, and
 ↪  types. Ensure that these details are used as-is in the function
 ↪  implementation.
2. **Analyze Question-Answer Pairs**: Examine these pairs to
 ↪  understand how questions map to function inputs and how answers
 ↪  should be derived from function outputs.
3. **Implement the Function**:
   - Use the tool-specified function name.
   - Define parameters exactly as specified in the tool document.
   - Implement logic to correctly derive answers for questions based
    ↪  on the input parameters.
   -  When parameters are assigned default values, Make sure that the
    ↪  function return value contains the complete given answer,
    ↪  i.e., the answer is a substring of the return value.
   - Ensure the function is capable of returning various responses,
    ↪  simulating additional return values if necessary.
4. **Error Handling**: Develop a comprehensive mechanism to return
 ↪  error messages for incorrect inputs or other issues, ensuring the
 ↪  function operates reliably in all scenarios.

# Output Format

The result should be output in JSON format, adhering to the following
 ↪  structure **without anything else**:
- "analysis": A detailed explanation of the function design,
 ↪  including reasoning for parameter choices and exception handling
 ↪  logic.
- "function": The specific implementation of the function, with code
 ↪  and comments that describe each component.

# Notes

- Match parameter names and types exactly with those specified in the
 ↪  tool document.
- Simulate additional return values as needed, based on the tool's
 ↪  documentation.
- Rely only on Python3's built-in libraries for all function
 ↪  implementations.
- Ensure comprehensive error handling to anticipate and manage
 ↪  potential issues effectively.

**Tool Document**:
{document}
```

```
**Question-Answer Pairs**:
{pairs}

**Output**:

'''
```

```
CODE_VERIFY = '''
Analyze the relationship between a given piece of function code and a
 ↪  problem to provide the appropriate function call to solve the
 ↪  problem, ensuring that all parameter values are derived from the
 ↪  problem.

# Steps

1. **Understand the Problem**: Read and comprehend the details of the
 ↪  given problem.
2. **Analyze the Code**: Examine the provided function code to
 ↪  ascertain how it addresses the problem.
3. **Confirm Code-to-Problem Suitability**: Determine if the function
 ↪  correctly solves the problem as described.
4. **Derive Function Call**: Craft a function call using the
 ↪  problem's specific details for parameter values.

# Output Format

Output the result in the following JSON format without any additional
 ↪  text:

- "analysis": A description analyzing how the given code relates to
 ↪  and addresses the problem.
- "call": The function call formatted as func(param="value"), using
 ↪  the problem's details for the parameter values.
  - If the fucntion can not used to solve the problem, return "call":
   ↪  null.

# Examples

**Example 1:**

- Input Problem: "Calculate the area of a rectangle with width=5 and
 ↪  height=10."
- Input Code: `def calculate_area(width, height): return width *
 ↪  height`

- JSON Output:
  - {{
    "analysis": "The function calculate_area computes the area of a
     ↪  rectangle by multiplying width and height. Given the problem,
     ↪  it appropriately solves it.",
    "call": "calculate_area(width=5, height=10)"
  }}

**Example 2:**

- Input Problem: "Find the square of the number 7."
- Input Code: `def square(num): return num * num`

- JSON Output:
```

```
  - {{
    "analysis": "The function square returns the square of the input
    ↪  number. It is suitable for finding the square of 7 as
    ↪  described in the problem.",
    "call": "square(num=7)"
  }}

# Notes

- Ensure the function call uses the exact parameter names as defined
↪  in the function code.
- The analysis should concisely justify the relationship between the
↪  function and the problem solution.

- Input Problem: {question}
- Input Code: {code}

- JSON Output:

'''
```

```
Qwen2_5_chat_template = '''
{%- if tools %}
    {{- '<|im_start|>system\n' }}
    {%- if messages[0].role == 'system' %}
        {{- messages[0].content + '\n\n' }}
    {%- endif %}
    {{- "# Tools\n\nYou may call one or more functions to assist with
    ↪  the user query.\n\nYou are provided with function signatures
    ↪  within <tools></tools> XML tags:\n<tools>" }}
    {%- for tool in tools %}
        {{- "\n" }}
        {{- tool | tojson }}
    {%- endfor %}
    {{- "\n</tools>\n\nFor each function call, return a json object
    ↪  with function name and arguments within
    ↪  <tool_call></tool_call> XML tags:\n<tool_call>\n{\"name\":
    ↪  <function-name>, \"arguments\":
    ↪  <args-json-object>}\n</tool_call><|im_end|>\n" }}
{%- else %}
    {%- if messages[0].role == 'system' %}
        {{- '<|im_start|>system\n' + messages[0].content +
        ↪  '<|im_end|>\n' }}
    {%- endif %}
{%- endif %}

{%- set ns = namespace(multi_step_tool=true,
↪  last_query_index=messages|length - 1) %}
{%- for message in messages[::-1] %}
    {%- set index = (messages|length - 1) - loop.index0 %}
    {%- if ns.multi_step_tool and message.role == "user" and
    ↪  message.content is string and
    ↪  not(message.content.startswith('<tool_response>') and
    ↪  message.content.endswith('</tool_response>')) %}
        {%- set ns.multi_step_tool = false %}
        {%- set ns.last_query_index = index %}
    {%- endif %}
{%- endfor %}

{%- for message in messages %}
    {%- if message.content is string %}
        {%- set content = message.content %}
    {%- else %}
        {%- set content = '' %}
    {%- endif %}

    {%- if (message.role == "user") or (message.role == "system" and
    ↪  not loop.first) %}
        {{- '<|im_start|>' + message.role + '\n' + content +
        ↪  '<|im_end|>\n' }}

    {%- elif message.role == "assistant" %}
        {%- set reasoning_content = '' %}
```

```
    {%- if message.reasoning_content is string %}
        {%- set reasoning_content = message.reasoning_content %}
    {%- else %}
        {%- if '</think>' in content %}
            {%- set reasoning_content =
            ↪  content.split('</think>')[0].rstrip('\n').split
            ↪  ('<think>')[-1]. lstrip('\n') %}
            {%- set content =
            ↪  content.split('</think>'[-1].lstrip('\n') %}
        {%- endif %}
    {%- endif %}

    {%- if loop.index0 > ns.last_query_index %}
        {%- if loop.last or (not loop.last and reasoning_content)
        ↪  %}
            {{- '<|im_start|>' + message.role + '\n<think>\n' +
            ↪  reasoning_content.strip('\n') + '\n</think>\n\n'
            ↪  + content.lstrip('\n') }}
        {%- else %}
            {{- '<|im_start|>' + message.role + '\n' + content }}
        {%- endif %}
    {%- else %}
        {{- '<|im_start|>' + message.role + '\n' + content }}
    {%- endif %}

    {%- if message.tool_calls %}
        {%- for tool_call in message.tool_calls %}
            {%- if (loop.first and content) or (not loop.first)
            ↪  %}
                {{- '\n' }}
            {%- endif %}
            {%- if tool_call.function %}
                {%- set tool_call = tool_call.function %}
            {%- endif %}
            {{- '<tool_call>\n{"name": "' }}
            {{- tool_call.name }}
            {{- '", "arguments": ' }}
            {%- if tool_call.arguments is string %}
                {{- tool_call.arguments }}
            {%- else %}
                {{- tool_call.arguments | tojson }}
            {%- endif %}
            {{- '}\n</tool_call>' }}
        {%- endfor %}
    {%- endif %}
    {{- '<|im_end|>\n' }}

{%- elif message.role == "tool" %}
    {%- if loop.first or (messages[loop.index0 - 1].role !=
    ↪  "tool") %}
        {{- '<|im_start|>user' }}
    {%- endif %}
```

```
        {{- '\n<tool_response>\n' + content + '\n</tool_response>' }}
        {%- if loop.last or (messages[loop.index0 + 1].role !=
        ↪  "tool") %}
            {{- '<|im_end|>\n' }}
        {%- endif %}
    {%- endif %}
{%- endfor %}

{%- if add_generation_prompt %}
    {{- '<|im_start|>assistant\n' }}
    {%- if enable_thinking is defined and enable_thinking is false %}
        {{- '<think>\n\n</think>\n\n' }}
    {%- endif %}
{%- endif %}
'''
```

```
Qwen3_non_reasoning_chat_template = '''
{%- if tools %}
    {{- '<|im_start|>system\n' }}
    {%- if messages[0].role == 'system' %}
        {{- messages[0].content + '\n\n' }}
    {%- endif %}
    {{- "# Tools\n\nYou may call one or more functions to assist with
    ↪  the user query.\n\nYou are provided with function signatures
    ↪  within <tools></tools> XML tags:\n<tools>" }}
    {%- for tool in tools %}
        {{- "\n" }}
        {{- tool | tojson }}
    {%- endfor %}
    {{- "\n</tools>\n\nFor each function call, return a json object
    ↪  with function name and arguments within
    ↪  <tool_call></tool_call> XML tags:\n<tool_call>\n{\"name\":
    ↪  <function-name>, \"arguments\":
    ↪  <args-json-object>}\n</tool_call><|im_end|>\n" }}
{%- else %}
    {%- if messages[0].role == 'system' %}
        {{- '<|im_start|>system\n' + messages[0].content +
        ↪  '<|im_end|>\n' }}
    {%- endif %}
{%- endif %}

{%- for message in messages %}
    {%- if message.content is string %}
        {%- set content = message.content %}
    {%- else %}
        {%- set content = '' %}
    {%- endif %}

    {%- if (message.role == "user") or (message.role == "system" and
    ↪  not loop.first) %}
        {{- '<|im_start|>' + message.role + '\n' + content +
        ↪  '<|im_end|>\n' }}

    {%- elif message.role == "assistant" %}
        {{- '<|im_start|>' + message.role + '\n' + content }}

        {%- if message.tool_calls %}
            {%- for tool_call in message.tool_calls %}
                {{- '\n' }}
                {%- if tool_call.function %}
                    {%- set tool_call = tool_call.function %}
                {%- endif %}
                {{- '<tool_call>\n{"name": "' }}
                {{- tool_call.name }}
                {{- '", "arguments": ' }}
                {%- if tool_call.arguments is string %}
                    {{- tool_call.arguments }}
                {%- else %}
```

```
                    {{- tool_call.arguments | tojson }}
                {%- endif %}
                {{- '}\n</tool_call>' }}
            {%- endfor %}
        {%- endif %}
        {{- '<|im_end|>\n' }}

    {%- elif message.role == "tool" %}
        {%- if loop.first or (messages[loop.index0 - 1].role !=
        ↪  "tool") %}
            {{- '<|im_start|>user' }}
        {%- endif %}
        {{- '\n<tool_response>\n' + content + '\n</tool_response>' }}
        {%- if loop.last or (messages[loop.index0 + 1].role !=
        ↪  "tool") %}
            {{- '<|im_end|>\n' }}
        {%- endif %}
    {%- endif %}
{%- endfor %}

{%- if add_generation_prompt %}
    {{- '<|im_start|>assistant\n<think>\n\n</think>\n\n' }}
{%- endif %}
'''
```

```
Qwen3_reasoning_chat_template = '''
{%- if tools %}
    {{- '<|im_start|>system\n' }}
    {%- if messages[0].role == 'system' %}
        {{- messages[0].content + '\n\n' }}
    {%- endif %}
    {{- "# Tools\n\nYou may call one or more functions to assist with
    ↪  the user query.\n\nYou are provided with function signatures
    ↪  within <tools></tools> XML tags:\n<tools>" }}
    {%- for tool in tools %}
        {{- "\n" }}
        {{- tool | tojson }}
    {%- endfor %}
    {{- "\n</tools>\n\nFor each function call, return a json object
    ↪  with function name and arguments within
    ↪  <tool_call></tool_call> XML tags:\n<tool_call>\n{\"name\":
    ↪  <function-name>, \"arguments\":
    ↪  <args-json-object>}\n</tool_call><|im_end|>\n" }}
{%- else %}
    {%- if messages[0].role == 'system' %}
        {{- '<|im_start|>system\n' + messages[0].content +
        ↪  '<|im_end|>\n' }}
    {%- endif %}
{%- endif %}

{%- set ns = namespace(multi_step_tool=true,
↪  last_query_index=messages|length - 1) %}
{%- for message in messages[::-1] %}
    {%- set index = (messages|length - 1) - loop.index0 %}
    {%- if ns.multi_step_tool and message.role == "user" and
    ↪  message.content is string and not
    ↪  (message.content.startswith('<tool_response>') and
    ↪  message.content.endswith('</tool_response>')) %}
        {%- set ns.multi_step_tool = false %}
        {%- set ns.last_query_index = index %}
    {%- endif %}
{%- endfor %}

{%- for message in messages %}
    {%- if message.content is string %}
        {%- set content = message.content %}
    {%- else %}
        {%- set content = '' %}
    {%- endif %}

    {%- if (message.role == "user") or (message.role == "system" and
    ↪  not loop.first) %}
        {{- '<|im_start|>' + message.role + '\n' + content +
        ↪  '<|im_end|>\n' }}

    {%- elif message.role == "assistant" %}
        {%- set reasoning_content = '' %}
```

```
        {%- if message.reasoning_content is string %}
            {%- set reasoning_content = message.reasoning_content %}
        {%- elif '</think>' in content %}
            {%- set reasoning_content =
            ↪  content.split('</think>')[0].rstrip('\n').split
            ↪  ('<think>')[-1]. lstrip('\n') %}
            {%- set content =
            ↪  content.split('</think>')[-1].lstrip('\n') %}
        {%- endif %}
        {{- '<|im_start|>' + message.role + '\n<think>\n' +
        ↪  reasoning_content.strip('\n') + '\n</think>\n\n' +
        ↪  content.lstrip('\n') }}

        {%- if message.tool_calls %}
            {%- for tool_call in message.tool_calls %}
                {{- '\n' }}
                {%- if tool_call.function %}
                    {%- set tool_call = tool_call.function %}
                {%- endif %}
                {{- '<tool_call>\n{"name": "' }}
                {{- tool_call.name }}
                {{- '", "arguments": ' }}
                {%- if tool_call.arguments is string %}
                    {{- tool_call.arguments }}
                {%- else %}
                    {{- tool_call.arguments | tojson }}
                {%- endif %}
                {{- '}\n</tool_call>' }}
            {%- endfor %}
        {%- endif %}
        {{- '<|im_end|>\n' }}

    {%- elif message.role == "tool" %}
        {%- if loop.first or (messages[loop.index0 - 1].role !=
        ↪  "tool") %}
            {{- '<|im_start|>user' }}
        {%- endif %}
        {{- '\n<tool_response>\n' + content + '\n</tool_response>' }}
        {%- if loop.last or (messages[loop.index0 + 1].role !=
        ↪  "tool") %}
            {{- '<|im_end|>\n' }}
        {%- endif %}
    {%- endif %}
{%- endfor %}

{%- if add_generation_prompt %}
    {{- '<|im_start|>assistant\n' }}
{%- endif %}
'''
```