TTPA: Token-level Tool-use Preference Alignment Training Framework with Fine-grained Evaluation

Anonymous ACL submission

Abstract

Existing tool-learning methods usually rely on supervised fine-tuning, they often overlook fine-grained optimization of internal tool call details, leading to limitations in preference alignment and error discrimination. To overcome these challenges, we propose **T**oken-level **T**ool-use **P**reference **A**lignment Training Framework (TTPA), a training paradigm for constructing token-level tool-use preference datasets that align LLMs with fine-grained preferences using a novel error-oriented scoring mechanism. TTPA first introduces reversed dataset construction, a method for creating high-quality, multi-turn tool-use datasets by reversing the generation flow. Additionally, we propose **T**oken-level **P**reference **S**ampling (TPS) to capture fine-grained preferences by modeling token-level differences during generation. To address biases in scoring, we introduce the **E**rror-oriented **S**coring **M**echanism (ESM), which quantifies tool-call errors and can be used as a training signal. Extensive experiments on three diverse benchmark datasets demonstrate that TTPA significantly improves tool-using performance while showing strong generalization ability across models and datasets. [1]

1 Introduction

Enabling **L**arge **L**anguage **M**odels (LLMs) (OpenAI, 2023; Touvron et al., 2023) to interact with external environments is critical for enhancing their ability to solve complex real-world problems through access to real-time information, such as web searches (Patil et al., 2024) and travel planning (Hao et al., 2024; Xie et al., 2024). As LLMs continue to evolve, integrating external tools is essential not only to address practical user needs but also to advance toward artificial general intelligence (Wang et al., 2023; Liu et al., 2023; Tian et al., 2024). Current approaches primarily employ **S**upervised **F**ine-**T**uning (SFT) to improve the tool-use capabilities of LLM (Qin et al., 2023b; Lin et al., 2024; Zhang et al., 2024; Tang et al., 2023; Schick et al., 2023). Recent studies also explore Reinforcement Learning (RL) for tool learning, such as TL-Training (Ye et al., 2024), which employs complex reward functions for proximal policy optimization (Schulman et al., 2017). Another approach leverages trajectory-level sampling to generate preference-based datasets for Direct Preference Optimization (DPO) (Rafailov et al., 2023). Although these RL-based methods offer a promising method for achieving preference alignment in tool use (Qin et al., 2024), they encounter two main challenges: (1) Existing methods often *overlook fine-grained preference discrepancies* within individual tool calls, where subtle token-level differences can determine the success or failure of the call. In highly structured outputs like tool calls, even a single token error can lead to complete failure, highlighting the necessity for more precise preference alignment. (2) Furthermore, existing preference data sampling methods typically rely on LLM-based or human evaluations, which may *introduce biases due to coarse-grained assessments* and ambiguous criteria. This often results in preference data with low discriminative quality and high noise levels, limiting the effectiveness of alignment strategies.

To overcome these two challenges, we propose **T**oken-level **T**ool-use **P**reference **A**lignment Training Framework (TTPA), a tool-use training paradigm that first constructs token-level preference datasets that align LLMs with fine-grained preferences, and then employs an error-oriented reward mechanism to train the model. Our proposed TTPA contains two main steps: (1) **P**reference **O**riented **T**ool-use **D**ataset **C**onstruction and (2) **E**rror-oriented **S**coring **M**echanism. We first propose a reversed data construction approach, which introduces a novel paradigm for creating multi-turn

[1] Code is available on Anonymous GitHub

tool-use datasets. Unlike conventional methods that start with queries, our approach reverses the process: we first leverage LLMs to generate a sequence of tool calls and a final answer within a predefined tool-using scenario. The query is then constructed based on the generated answer. This reversed strategy ensures that every query is inherently answerable and eliminates data leakage risks, as the query is derived from the scenario and answer rather than predefined inputs.

To capture the fine-grained preference in the tool calls, we propose **T**oken-level **P**reference **S**ampling (TPS). Unlike trajectory-level methods that incorporate complete tool-calling sequences, our approach explicitly models token-level preferences by sampling top-k candidate tokens from the probability distribution during tool-call generation by LLM. When training the tool-use LLM, existing models employ LLMs to grade the outputs as the training signal which usually introduces biases caused by coarse-grained evaluation and ambiguous criteria (Nath et al., 2025). Thus, we propose the **E**rror-oriented **S**coring **M**echanism, which defines a taxonomy of tool-call errors. And then we use it to construct a preference alignment dataset and fine-tune the LLM . Extensive experiments on three benchmark datasets show that TTPA notably improves tool selection, parameter filling, and return value parsing capabilities. Moreover, the model fine-tuned with TTPA demonstrates strong generalization and transferability across datasets, enhancing the reliability and applicability of LLMs in real-world applications.

In summary, our contributions are as follows:

• We propose **T**oken-level **T**ool-use **P**reference **A**lignment Training Framework (TTPA), a novel tool-use training paradigm that aligns the LLM with fine-grained token-level preference to avoid the tool-call error.

• We introduce the **P**reference **O**riented **T**ool-use **D**ataset **C**onstruction, which employs a reversed data construction method and construct fine-grained preference data.

• We propose the **E**rror-oriented **S**coring **M**echanism (ESM), which captures fine-grained differences between answers, enabling precise alignment of LLM.

• Experimental results demonstrate that TTPA significantly improves tool-use capabilities on three diverse benchmark datasets, and shows strong generalization across models and datasets.

2 Related work

Tool Learning. Tool learning enhances LLMs by integrating external tools, enabling them to select tools, generate parameters, and parse results to respond to user queries (Qin et al., 2023a; Li et al., 2023; Huang et al., 2023; Shi et al., 2023). Approaches include tuning-free methods, which use in-context learning or algorithmic design (Yao et al., 2023; Shi et al., 2024b; Huang et al., 2024; Zhu et al., 2025), and tuning-based methods, which fine-tune on tool-use datasets (Wu et al., 2024; Kong et al., 2024; Gao et al., 2024). Tuning-free methods are often limited by the foundation model's capabilities, while tuning-based methods face challenges with noisy data. Our framework addresses this by employing Reversed Dataset Construction and Token-level Preference Sampling to produce high-quality, low-noise datasets, ensuring better alignment with tool-use tasks and addressing fine-grained discrepancies in tool calls. Additionally, our approach introduces an error-oriented scoring mechanism to refine the alignment process and improve model robustness in complex scenarios.

Tool-Use Datasets. Tool learning has driven the creation of datasets to improve LLMs' tool-use capabilities (Patil et al., 2023; Wang et al., 2024a; Gao et al., 2024). ToolBench (Qin et al., 2023b) leverages LLMs to compile large datasets, while APIGen (Liu et al., 2024b) uses an automated pipeline to generate diverse datasets across multiple API categories. ToolACE (Liu et al., 2024a) further advances this by integrating tool synthesis and dialogue generation, enhancing dataset diversity and complexity. However, these datasets often suffer from noise, single-turn limitations, or high resource costs, and few address the growing need for preference-based datasets. Our framework uses Reversed Dataset Construction and Token-level Preference Sampling to construct high-quality preference datasets, aligning token-level tool-use preferences and improving fine-grained alignment for structured outputs, ensuring better generalization across diverse tool-use scenarios.

3 Method

3.1 Overview

In this section, we present the details of **T**oken-level **T**ool-use **P**reference **A**lignment Training Framework (TTPA). An overview of TTPA is illustrated in Figure 1, which contains three key components: (1) First, we introduce the **R**eversed **D**ataset **C**onstruction, which generates a reliable and non-leaked raw conventional instruction dataset like any other public dataset, serving as the foundation for the preference dataset. (2) Next, we describe our **T**oken-level **P**reference **S**ampling strategy, which constructs *Preferred & Dispreferred* pairs by calculating scores through the fine-grained **E**rror-oriented **S**coring **M**echanism. (3) Finally, we introduce the **E**rror-oriented **S**coring **M**echanism which is designed to capture token-level preferences.

3.2 Reversed Dataset Construction

In existing tool-use datasets, the generated queries may explicitly reveal information about the tools or parameters involved (Qin et al., 2023b). However, in real-world scenarios, user queries typically do not explicitly specify the tools to be called or the input parameters. This discrepancy creates a gap between the dataset and real-world applications, ultimately affecting the model's performance in practical settings. Unlike traditional approaches (Qin et al., 2023b) that guide LLMs to first generate a query Q and then solve it, which may result in unsolvable or overly ambiguous queries, we propose a novel method that constructs tool-use training data by deriving queries from answers. To address these issues, we propose the **R**eversed **D**ataset **C**onstruction method to construct a tool-use dataset.

First, we use a candidate tool set T_{can} as input and then prompt the generator $\mathscr{G}$ to construct three items:

$$\{S, T_{\text{use}}, \text{Cons}\} = \mathscr{G}(P_{\text{S}}, T_{\text{can}}), \tag{1}$$

where P_S denotes the prompt and the outputs are: (1) A tool-use scenario description S which is a short sentence to describe this tool-use application scenario. (2) A toolset $T_{\text{use}} = \{t_1, t_2, \cdots, t_N\}$ with N tools is selected according to the task requirement in the scenario, which should be used in the scenario *S*. (3) Some constraint Cons of the scenario *S* to restrict the solution space.

Next, our goal is to generate an answer *A* based on the tool-use application scenario *S*. We simulate the task-solving process by iteratively selecting and calling the tools in T_{use}. Specifically, in each tool calling step, we predict the tool used in the i-th step t^i_{call} according to these inputs and obtain the output t^i_{res} of the tool t^i_{call}.

$$t^i_{\text{call}} = \mathscr{G}(P_A, S, T_{\text{use}}, Cons, M^{i-1}), \tag{2}$$

$$\text{where} \quad M^{i-1} = \bigcup_{j}^{i-1} \{t^j_{\text{call}}, t^j_{\text{res}}\}.$$

$$t^i_{\text{res}} = \text{Call}(t^i_{\text{call}}), \tag{3}$$

where M^{i-1} presents the historical tool calls and results up to $i-1$ step, and P_A denotes the answer generation prompt. After multiple rounds of tool interactions, the generator $\mathscr{G}$ obtains a series of results returned by the tools, and then we generate the answer A according to these inputs:

$$A = \mathscr{G}(P_{\text{A}}, S, T_{\text{use}}, Cons, M), \tag{4}$$

where M denotes the previous tool calls and results. Finally, we instruct the generator $\mathscr{G}$ to generate a query Q:

$$Q = \mathscr{G}(P_Q, S, \text{Cons}, A, T_{\text{calls}}). \tag{5}$$

Since the queries are derived from answers, each query in this dataset is guaranteed to have a valid solution. Furthermore, the queries, answers, and associated tool results are highly correlated, ensuring that solving the queries necessitates the use of tools. This design significantly reduces noise in the dataset, resulting in higher data quality.

3.3 Token-level Preference Sampling

Since the trajectory-level sampling method (CHEN et al., 2024), which aligns preferences at a macro level by capturing the overall learning path, usually fails to account for fine-grained distinctions within individual trajectories. To tackle this problem, we propose the **T**oken-level **P**reference **S**ampling (TPS) strategy for **D**irect **P**reference **O**ptimization (DPO). For brevity, we denote by M^i_{pre} the set of tool calls and their corresponding return values prior to the i-th tool call:

$$M^i_{\text{pre}} = \{t^1_{\text{call}}, t^1_{\text{res}}, \ldots, t^{i-1}_{\text{call}}, t^{i-1}_{\text{res}}\}. \tag{6}$$

To construct a preference dataset more suitable for training the tool learning model $\mathscr{L}$, we build the preference dataset by sampling from the outputs of the tool learning model $\mathscr{L}$:

$$P_{\text{pred}} = \mathscr{L}(Q, T_{\text{use}}, M^i_{\text{pre}}), \tag{7}$$

where P_{pred} denotes the predicted probability distribution over *tool calls* generated by the tool learning model $\mathscr{L}$ for the *i*-th step.

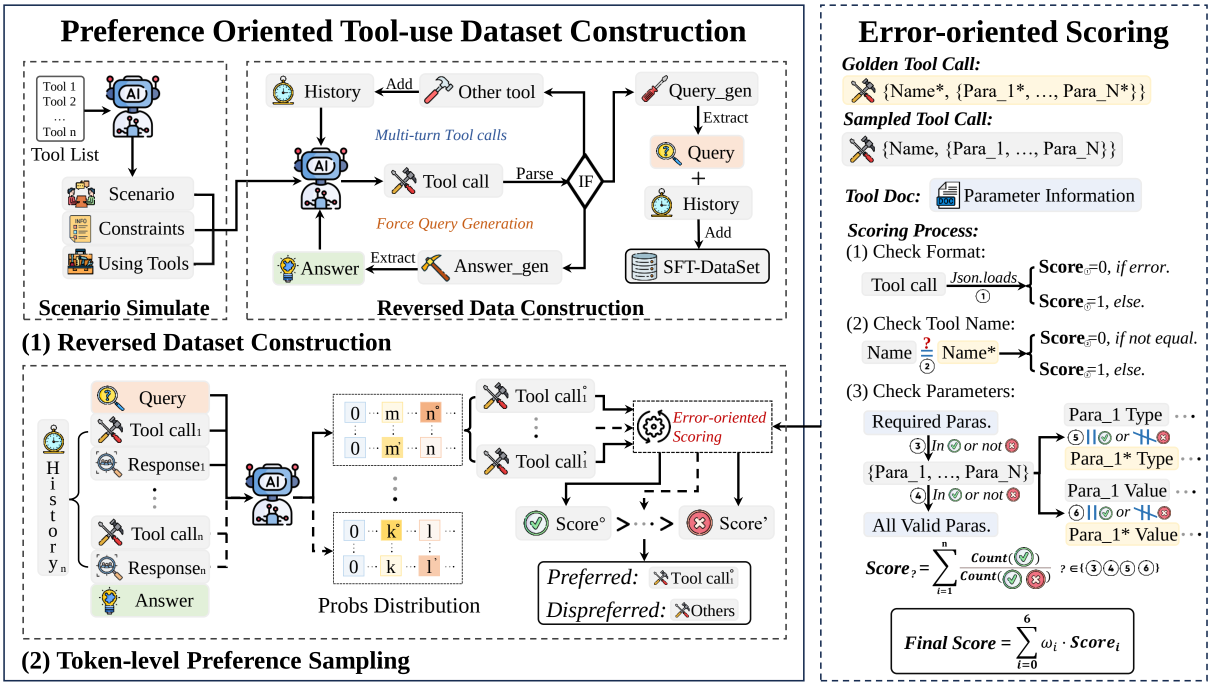

Figure 1: The overall framework of our work, which mainly consists of Preference Oriented Tool-use Dataset Construction and Error-oriented Scoring Mechanism.

During the token-by-token generation process of tool learning model $\mathscr{L}$, the token probability distribution P_{pred} over the entire vocabulary is computed before each token is generated. During sampling, candidate tokens are selected from the top-ranked tokens in P_{pred}. However, the probability gap between the top-ranked tokens is not always significant, and the probabilities of the top-ranked tokens are very close. This close probabilities' distribution creates ambiguity during decoding, as different decoding strategies may randomly select different high-probability tokens. Such randomness is particularly problematic for structured and fixed outputs like tool calls, where even a single incorrect token can lead to the failure of the entire tool call. Therefore, we use the uncertainty in token probabilities as a sampling criterion, perturbing only a small number of tokens at a time to simulate the uncertain sampling behavior of LLMs during the decoding phase:

$$C^K_{\text{sam}} \sim P_{\text{pred}}\mathbb{I}(\text{Dist} < \epsilon), \tag{8}$$

$$\text{where} \quad \text{Dist} = p_{r_1} - p_{r_j}, \tag{9}$$

where C^K_{sam} denotes K-times tool call sampling results in the condition of the distance *Dist* between *rank-j* token's probability p_{r_j} and *rank-1* token's probability p_{r_1} smaller than the predefined hyper-parameter ϵ, the value of K is dynamically determined based on the specific probability. Unlike deterministic decoding methods (Shi et al., 2024a), which often produce repetitive or suboptimal results, our approach introduces controlled randomness by perturbing a small number of tokens based on their uncertainty.

Next, we compute the score ψ_i for each sampled *tool call* $c^i_{\text{sam}} \in C^K_{\text{sam}}$:

$$\psi_i = \mathscr{F}(c^i_{\text{sam}}), \tag{10}$$

where $\mathscr{F}$ is the scoring mechanism that can capture fine-grained errors that may occur during tool calls, enabling precise alignment of model preferences, and the detail for this mechanism will be introduced in § 3.4. Finally, the sample with the highest score ψ is selected as the *Preferred Answer*, while the remaining samples are designated as *Dispreferred Answers*.

3.4 Error-oriented Scoring Mechanism

Existing tool learning methods usually employ LLM-based evaluation or human evaluation to assess the quality of generated tool calls, and then use this signal to optimize the model parameters. In this paper, we design an error-oriented scoring mechanism $\mathscr{F}$ that can capture fine-grained errors that may occur during tool calls. For tool learning tasks, since tool calls are structured representations, we propose a taxonomy for the tool-call errors. For a tool call result t_{call}, the scoring function δ is designed to identify whether the call contains errors and to classify these errors into specific error types:

$$\delta^{e_i}(t_{\text{call}}) = \begin{cases} 0, & \text{if } e_i \text{ detected.} \\ 1, & \text{if } e_i \text{ not detected.} \end{cases} \tag{11}$$

where e_i denotes a specific error type (*e.g.,* format errors and tool name errors).

However, since different tools may have varying numbers of parameters, simply matching the predicted parameters with the ground-truth parameters could result in coarse-grained outcomes. Therefore, we perform a detailed validation on each parameter output by the model, including type errors and value errors. In our evaluation method, each parameter is assigned a score, and the final scores for parameter type errors and parameter value errors are obtained by taking the weighted average of all parameter scores:

$$\delta^{e_i}(t_{\text{call}}) = \frac{1}{X}\sum_{j}^{X}\gamma(v_j), \tag{12}$$

where $\gamma(v_j)$ denotes a similar function to score each parameter v of the X parameters generated by tool learning model $\mathscr{L}$, which can be represented as:

$$\gamma(v_j) = \begin{cases} 0, & \text{if } v_j \text{ not correct.} \\ 1, & \text{if } v_j \text{ correct.} \end{cases}. \tag{13}$$

After the scores for all error types are computed, we obtain the final score for the tool call by weighted sum the scores of all types of errors detecting:

$$\mathscr{F}(t_{\text{call}}) = \sum_{i}^{H}\omega_i \cdot \delta^{e_i}(t_{\text{call}}), \tag{14}$$

where ω_i denotes the hyper-parameter weight of the type of error e_i, $\delta^{e_i}(t_{\text{call}})$ denotes the score of each type of error and H denotes the total number of error types. This scoring mechanism can be utilized to generate a preference-aligned dataset, which is subsequently employed for training tool learning models using the DPO method.

4 Experimental Setup

4.1 Implementation Details

To evaluate the effectiveness of the proposed TTPA, we initially employ Reversed Data Construction and Token-level Preference Data Sampling techniques to generate 3895 instruction data instances and 8550 preference data pairs, utilizing 114 specialized apis for data generation and processing. During the data generation phase, we leverage state-of-the-art language models, specifically GPT-4 Mini and GPT-4 (OpenAI, 2023), as our primary generators $\mathscr{G}$ to ensure both the quality and validity of the synthesized data. Following the data generation phase, we employ Qwen2.5-7B-Instruct (Qwen et al., 2025) as the tool learning model $\mathscr{L}$ and conduct fine-tuning procedures on the generated dataset to optimize its performance.

4.2 Baseline

We conduct a comprehensive comparison between our proposed TTPA and several state-of-the-art baselines in tool use, including: (1) `GPT-4o-mini`, developed by OpenAI, which exhibits exceptional performance in tool-use. (2) `Hammer2.0-7b` (Lin et al., 2024), a state-of-the-art tool learning model, demonstrates exceptional function calling capabilities, particularly excelling in robustness during tool call. (3) `ToolACE-8B` (Liu et al., 2024a), an advanced tool learning model, is specifically trained on coherent dialogue-based tool use datasets, endowing it with robust capabilities for tool utilization in multi-turn conversational. (4) `xLAM-7b-r` (Liu et al., 2024b), an advanced large language model designed to enhance decision-making and translate user intentions into executable actions that interact with the world, which training on 60k single-turn tool-use dataset.

4.3 Dataset & Metric

We evaluate the tool learning model fine-tuned with TTPA on two commonly-used benchmarks and our proposed testset. The statistics of these datasets are shown in Table 2. We first use the subset of widely-used ToolBench (Qin et al., 2023b) benchmark, including *I1-instruction* and *I1-tool*. For evaluation, we employ the *Pass Rate* metric, which serves as an intuitive measure of tool learning LLMs' capability in accurately selecting appropriate tools and generating corresponding parameters by the model within a constrained number of inference steps. Moreover, we employ the Berkeley Function-Calling Benchmark (BFCL) (Patil et al., 2024), which covers complex scenarios such as multiple tool use. We utilize five subsets from the BFCL, comprising a total of 1,929 instances for our test set. In the evaluation framework, BFCL primarily assesses LLMs based on Abstract Syntax Tree (AST) Evaluation. This evaluation measures the syntac-

Models	Vanilla	QS	QL	TS	TE	TCE
I1-instruction						
GPT-4o-mini	82.0%	80.0%	83.5%	84.0%	81.5%	81.0%
Hammer2.0-7b	60.0%	56.0%	54.5%	58.0%	51.5%	53.0%
xLAM-7b-r	77.5%	78.5%	73.5%	79.5%	75.5%	73.0%
ToolACE-8B	77.0%	75.5%	78.5%	74.0%	72.0%	72.0%
TTPA (Qwen)	**86.0**%	**88.5**%	**84.5**%	**87.5**%	**86.0**%	**83.5**%
I1-tool						
GPT-4o-mini	**85.5**%	83.5%	80.0%	81.5%	83.0%	82.0%
Hammer2.0-7b	62.0%	66.0%	56.0%	68.5%	51.0%	51.0%
xLAM-7b-r	77.5%	77.0%	77.0%	73.5%	71.0%	69.5%
ToolACE-8B	76.0%	77.5%	**86.0**%	77.5%	76.0%	76.0%
TTPA (Qwen)	85.0%	**84.0**%	82.0%	**81.5**%	**83.0**%	**83.5**%

Table 1: The results of evaluation on various ToolBench subsets. The dataset abbreviations correspond to specific modifications: (1) Vanilla represents the original ToolBench dataset; (2) Query Shorten denotes the version with condensed queries for increased information density; (3) Query Lengthen indicates extended queries with additional information, resulting in sparser key information distribution; (4) Tools Shuffle refers to the variant with randomized tool candidate ordering; (5) Tools Expand (Intra-category) represents the expanded toolset within the same category; and (6) Tools Expand (Cross-category) indicates the expanded toolset across different categories.**Bold** values represent the highest performance for the models evaluated.

Attributes	ToolBench	BFCL	Ours
Subsets	12	5	1
Amount	2400	1929	385
APIs	1543	1100	114
Avg. APIs	5.06	1	5.56

Table 2: Statistics of the experimental datasets. APIs presents the total number of using APIs in the entire dataset, and Avg. APIs presents the average number of tool-calls per individual case.

tic correctness of generated tool calls by verifying their alignment with predefined tool documentation in terms of structure and parameters. And we also employ our testset where we randomly split 10% of the generated data (with 385 samples) for testing. In the testing process, we employ the error-oriented scoring mechanism as the evaluation metric, enabling a fine-grained assessment of tool calls.

5 Experimental Result

5.1 Overall Performance

To assess the effectiveness of our proposed TTPA, we conducted a comprehensive comparison of our model with several strong baseline models across three diverse datasets. The results are shown in Table 1, Table 3, and Table 4 for Toolbench, BFCL, and Our testset, respectively.

ToolBench The findings in ToolBench validate the effectiveness of training on tool-use datasets, revealing that models with merely 7-8 billion parameters can achieve comparable or even superior performance to state-of-the-art GPT-4o-mini in some subsets. This highlights the critical role of domain-specific fine-tuning in enhancing the tool-use capability of LLMs. Our TTPA outperforms the baselines in most scenarios, demonstrating the generalizability of our approach. However, an exception is observed in the QL sub-dataset under the I1-tool dataset, where ToolACE-8B achieves better performance. This discrepancy can likely be attributed to the fact that ToolACE incorporates extensive dialogue information during its training process, enabling it to handle long queries more effectively. Moreover, due to the long-context training data derived from a long candidate tool list, models are required to select the correct tool in more complex scenarios. Consequently, our model exhibits higher robustness across five out of six sub-datasets. In contrast to other models, where performance fluctuations exceed 5% even 10%, our model maintains a pass rate variation of less than 2%. The exception observed in the TCE sub-dataset, where performance declines, is likely due to the crossed expansion of the candidate tool list, which indicates that the model must first identify the appropriate sub-toolsets category before selecting the correct tool within that subset. Due to the lack of sufficient training data for this specific challenge, most mod-

Models	Multiple(live)	Simple(live)	Multiple	Simple	Relevance(live)
GPT-4o-mini	76.3%	77.1%	90.0%	90.5%	77.8%
Hammer2.0-7b	75.0%	67.4%	93.5%	95.2%	83.3%
xLAM-7b-r	**75.4%**	73.6%	95.0%	92.2%	**100.0%**
ToolACE-8B	75.2%	78.2%	**95.5%**	95.0%	94.4%
TTPA (Qwen)	71.7%	**79.5%**	93.0%	**95.5%**	94.5%

Table 3: Accuracy performance on the BFCL subsets. *Multiple* and *Simple* denote that the LLMs are provided multiple tools and one tool, respectively. *live* distinguishes itself from other datasets in the same category. **Bold** values represent the highest performance for the models evaluated.

Models	Name	Para.	Content
GPT-4o-mini	43.0%	70.3%	64.6%
Hammer2.0-7b	33.9%	67.3%	59.7%
xLAM-7b-r	39.6%	71.1%	63.1%
ToolACE-8B	31.7%	62.7%	51.1%
TTPA (Qwen)	**57.8%**	**81.3%**	**74.2%**

Table 4: Results on our testset. *Name*, *Para.* and *Content* denote the tool calls' accuracy of tool selection, parameters choosing, and parameters content filling, respectively. **Bold** values represent the highest performance for the models evaluated.

Error Types	Example	Reason
Format	`{······}`	Missing a "}".
Wrong tool name	`{"name":"tool",···}`	Wrong tool name "tool", correct "function".
Missing required para.	`{···,"paras.":{"year":2025,···}}`	Missing required para. "year".
Wrong para. name	`{···,"paras.":{"years":2025,···}}`	Wrong para. "years", correct "year".
Wrong para. type	`{···,"paras.":{"year":"2025",···}}`	Wrong para. type "string", correct "int".
Wrong para. value	`{···,"paras.":{"year":2036,···}}`	The value of "year" should be earlier than current year.

Figure 2: Error types of tool calls. *Example* column presents the examples of different error types. *Reason* column presents the reason why the example failed.

els perform worse on this dataset compared to their performance on the vanilla dataset. Nevertheless, our model still surpasses the baselines, achieving the best performance.

BFCL The results on BFCL demonstrate that the SOTA baseline models have achieved remarkable performance, particularly on the multiple and simple subsets, where they attain accuracy rates exceeding 90%. Notably, our fine-tuned model demonstrates comparable performance to these existing approaches, reaching SOTA performance levels. However, we identify a potential limitation in the BFCL evaluation system: its design may introduce bias during assessment since the number of solutions included for a specific case is fewer than the actual possible solutions. This limitation could lead to two main issues: (1) correct tool calls being misclassified as false, thereby reducing accuracy metrics, and (2) potential favoritism toward models trained on specific datasets that the datas' distribution is similar to the BFCL' data. These factors may partially explain why our model shows slightly inferior performance compared to SOTA models on certain subsets. More detailed case studies can be found in the Appendix A.1.

Our Testset Moreover, on our custom test set, our fine-tuned model outperforms existing advanced tool learning models across three critical aspects that show the capability of tool-use: tool name selection, parameters choosing, and parameters' value filling. Specifically, our model achieves accuracies of 57.8%, 81.3%, and 74.2%, respectively, representing at least an average improvement of 11.8% compared to the baseline advanced models.

All results on these test sets show the effectiveness of our proposed TTPA, which can enhance the LLMs' capability of tool-use.

5.2 Error Type Analysis

In tool-use tasks, LLM errors can be classified into three main categories of six types (Figure 2) (Dathathri et al., 2020; Ye et al., 2024). Analyzing these errors provides insights for optimizing LLMs' tool-use capabilities. The first category is format errors, where LLMs must generate machine-parsable tool calls, requiring strict adherence to correct output formats. The second category involves tool selection errors, as LLMs need to choose the most appropriate tool based on task requirements and a thorough understanding of each tool's functionality. The final category concerns parameter errors, which include missing required parameters, invalid parameter types, or values that significantly deviate from the golden references, particularly for parameters involving natural language text.

These errors reflect LLMs' capabilities in three dimensions: (1) instruction following (structured outputs), (2) document comprehension (tool selection), and (3) text generation (parameter filling). This analysis highlights LLMs' limitations and guides targeted improvements in tool-use tasks.

Dataset	Base Model	TTPA Model
ToolBench		
-I1-inst.(avg.)	46.3%	**86.0%**
-I1-tool(avg.)	51.5%	**83.2%**
BFCL		
-Multiple(avg.)	**83.3%**	82.4%
-Simple(avg.)	84.2%	**87.5%**
-Relevance	77.8%	**94.5%**
Ours		
-Testset(avg.)	43.3%	**71.1%**

Table 5: Ablation study. We employ Qwen2.5-7B-Instruct as base model, finetuning with TTPA. *avg.* presents the average accuracy across all subsets of the corresponding category or different evaluation aspects.

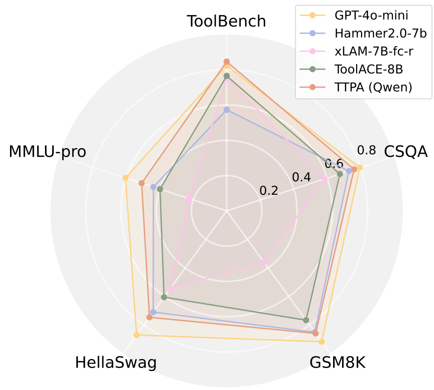

Figure 3: The results of evaluation on the general datasets.

5.3 Ablation Study

To evaluate the effectiveness of our proposed **T**oken-level **T**ool-use **P**reference **A**lignment Training Framework (TTPA) in enhancing the tool-use capabilities of LLMs, we conducted an ablation study comparing the tool-use performance of the base model across various scenarios before and after TTPA remarkably enhances the tool-use capabilities of LLMs. Specifically, we observed substantial improvements across all three benchmark datasets, with performance gains reaching up to 39.7%. These findings suggest that constructing token-level preference datasets for model fine-tuning enables more granular alignment with correct tool calls while identifying suboptimal or erroneous tool calls, thereby substantially improving tool-use performance.

5.4 General Performance

To comprehensively evaluate the impact of TTPA on the general capabilities of LLMs, we conduct experiments across multiple benchmarks that assess diverse cognitive abilities: MMLU-pro (Wang et al., 2024b) fro knowledge mastery, HellaSwag (Zellers et al., 2019) for commonsense reasoning, GSM8K (Cobbe et al., 2021) for mathematical problem-solving, CommonSenseQA (Talmor et al., 2019) for conceptual understanding, and ToolBench for tool-usage. The results, presented in Figure 3, demonstrate that the model fine-tuned with TTPA achieved comparable tool-use capabilities to the state-of-the-art GPT-4o-mini model while maintaining competitive performance across other general benchmarks. Furthermore, our analysis reveals that the model exhibits robust generalization capabilities across different domains, suggesting the effectiveness of the TTPA fine-tuning approach in both enhancing specialized and maintaining general-purpose performance.

6 Conclusion

In this paper, we present **T**oken-level **T**ool-use **P**reference **A**lignment Training Framework (TTPA), an automated method for constructing high-quality tool-use preference datasets to enhance the tool-use capability of large language models. The proposed TTPA employs a novel Preference Oriented Tool-use Dataset Construction, which incorporates two key components: (1) Reversed Data Construction for generating diverse tool-use dataset, and (2) Token-level Preference Sampling for capturing token-level preference. Additionally, we develop an Error-oriented Scoring Mechanism that enables precise alignment of LLMs with fine-grained user preferences during tool-usage. Experiment results demonstrate that the tool learning model fine-tuned with TTPA can achieve state-of-the-art performance, thereby advancing the field of tool usage in large language models (LLMs).

Limitations

The main limitation is that conducting fine-grained token-level preference sampling may lead to an increase in computational complexity, requiring higher computational resources and extending the overall training time. In future work, we plan to integrate efficient inference methods with our approach to enhance sampling efficiency. Additionally, our training data is based on a predefined static set of tools, whereas in practical applications, the external environment is dynamically changing. The model's adaptability in dynamic environments still requires further research and validation. We aim to construct a dynamic tool library and extend our method to this dynamic setting, further improving the model's tool-use capabilities in dynamic environments.

Ethical Considerations

The research conducted in this paper centers on investigating the effectiveness of fine-grained aligning LLMs for tool-usage. Our work systematically benchmarks LLMs under various real-world scenarios and evaluates their performance.

In the process of conducting this research, we have adhered to ethical standards to ensure the integrity and validity of our work. All the tasks as well as tools used in our experiment were obtained from existing benchmarks and public open resources, thus ensuring a high level of transparency and reproducibility in our experimental procedure.

To minimize potential bias and promote fairness, we use the prompts following existing works, which are publicly accessible and freely available. We have made every effort to ensure that our research does not harm individuals or groups, nor does it involve any form of deception or potential misuse of information.

References

SIJIA CHEN, Yibo Wang, Yi-Feng Wu, Qingguo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. In *Advances in Neural Information Processing Systems*.

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. *arXiv*.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations: ICLR*.

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

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2024. Large language models can plan your travels rigorously with formal verification tools. *arXiv preprint arXiv:2404.11891*.

Chengrui Huang, Zhengliang Shi, Yuntao Wen, Xiuying Chen, Peng Han, Shen Gao, and Shuo Shang. 2024. What affects the stability of tool learning? an empirical study on the robustness of tool learning frameworks. *arXiv*.

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

Yilun Kong, Jingqing Ruan, YiHong Chen, Bin Zhang, Tianpeng Bao, Shi Shiwei, du Guo Qing, Xiaoru Hu, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, and Xueqian Wang. 2024. TPTU-v2: Boosting task planning and tool usage of large language model-based agents in real-world industry systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Association for Computational Linguistics: EMNLP*.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. 2024a. Toolace: Winning the points of llm function calling. In *International Conference on Learning Representations: ICLR*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen

Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh R N, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024b. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*.

Vaskar Nath, Pranav Raja, Claire Yoon, and Sean Hendryx. 2025. Toolcomp: A multi-tool reasoning & process supervision benchmark. *arXiv*.

OpenAI OpenAI. 2023. Gpt-4 technical report.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, 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, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Guoliang Li, Zhiyuan Liu, and Maosong Sun. 2024. Tool learning with foundation models. In *ACM Comput. Surv.*

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023b. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *International Conference on Learning Representations: ICLR*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *arXiv*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *Neural Information Processing Systems: NeurIPS*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. 2024a. A thorough examination of decoding methods in the era of llms. *arXiv*.

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Zhumin Chen, Suzan Verberne, and Zhaochun Ren. 2024b. Chain of tools: Large language model is an automatic multi-tool learner. *ArXiv*.

Zhengliang Shi, Shen Gao, Zhen Zhang, Xiuying Chen, Zhumin Chen, Pengjie Ren, and Zhaochun Ren. 2023. Towards a unified framework for reference retrieval and related work generation. In *Association for Computational Linguistics: EMNLP*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*.

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. *arXiv preprint arXiv:2306.05301*.

Shubo Tian, Qiao Jin, Lana Yeganova, Po-Ting Lai, Qingqing Zhu, Xiuying Chen, Yifan Yang, Qingyu Chen, Won Kim, Donald C Comeau, et al. 2024. Opportunities and challenges for chatgpt and large language models in biomedicine and health. In *Briefings in Bioinformatics*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024a. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *International Conference on Learning Representations: ICLR*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024b. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv*.

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 2024. ToolPlanner: A tool augmented LLM for multi granularity instructions with path planning and feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations: ICLR*.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Wei Zhang, Yi Zhang, Li Zhu, Qianghuai Jia, Feijun Jiang, Hongcheng Guo, Zhoujun Li, and Mengping Zhou. 2024. Adc: Enhancing function calling via adversarial datasets and code line-level feedback.

Dongsheng Zhu, Weixian Shi, Zhengliang Shi, Zhaochun Ren, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. 2025. Divide-then-aggregate: An efficient tool learning method via parallel tool invocation. *arXiv*.

A Appendix

A.1 Case Study

A.1.1 BFCL

Figure 4 shows one case in the evaluation process of Multiple (live) subset of BFCL datasets, which TTPA (Qwen) failed while xLAM-7b-r success due to the limitation of the evaluate system of BFCL. As shown in Figure 4, the correct function *get_tesco_locations* has three acceptable parameters, where the parameters *radius* and *limit* are optional and not specified. But the golden answer just contains limited valid answers, such that TTPA (Qwen)'s output is evaluated as failure although it generates the correct API name and required parameters (including the parameter's name, type, and value).

A.2 Training Details

The hyper-parameters of the training process are illustrated in Table 6

A.3 Prompt Templates

The prompts we designed are listed below:

A.3.1 Reversed Dataset Construction

Prompt of **Scenario Simulation:**

Given the following tools, simulate a scenario where these tools are used in a real-world scenario.
You DO NOT need to actually use the tools, just simulate the scenario based on the information provided by the tools.
Your goal is to simulate a realistic scenario that involves multiple turns and multiple tools to help another answerer to answer the implicit question asked by a asker.

When simulating the scenario, consider the following:
1. The scenario should be as realistic as possible and should involve multiple turns (at least two tools).
2. The scenario should be related to the tools provided.

IMPORTANT: The scenario you simulate CAN NOT contain any explicit questions.
You SHOULD only state the scenario.
The scenario you simulate CAN NOT contain any tool name in the tools above.
You SHOULD keep the scenario as realistic as possible.

YOUR OUTPUT CONTAINS:
scenario: str, the scenario you simulated, it should be a few short words. Also, it should not be a question or instruction. It is just a statement about the scenario.

additional_information: list[str], any information you want to provide about the scenario that may help the answerer to understand the scenario better, at least 4, at most 7. Such as the time, the location, the people involved, etc.

tools: list[str], the tools' name you think are related to the scenario, you should choose the tools from the tools above. And the number of tools should be at least 7, at most 10.
There are the tools you can choose:
{tools}

Prompt of **Answer Generation:**

You are a data scientist tasked with generating questions to extract specific information from a given dataset.
Imagine that there is a asker, you should answer the asker's questions based on the tool calls.
But there is no explicit question, you need to answer the implicit question that the asker may have.

There are some Steps you can follow:
Steps:
1. Choose an appropriate tool that you believe can help generate the questions.
2. call the selected tool to obtain the tool calls.
3. If the tool calls are insufficient to generate the questions, select another tool and repeat the process.
4. Once you have gathered enough information, call the Answer_gen tool to generate an answer based on the tool calls.
5. If there are errors, such as the tool returns invalid information or the tool call failed, call the **Restart** tool to restart.

Rules:
1. You can choose only one tool at a time.
2. The task must involve multiple turns (at least two tools).
3. Simulate a realistic scenario in the Additional Information section.

Additional Information:
{add_info}

Note:
1. Adapt it to your role and make the task as complex and realistic as possible.
2. You should chose the tools related to the scenarios {scene} and the information provided.

Prompt of **Query Generation:**

Imagine that there is a answerer. The answerer answer a question by calling some tools.

But there is no explicit question, you need to guess the implicit question that the answerer may answer from the scenario and answer, tool calls given by the answerer.

Remember that the implicit question should be closely related to the tool calls and the final answer.

But if the answer does not give a clear answer because the tool calls failed, you should guess the implicit question as if the tool calls were successful.
Remember that the question should contains the key information that solve the task should be used, such as the date, the location, the people involved, the data to calculate, etc.

RULES:
1. The question should be designed such that the provided answer is the solution, and the sequence of tool calls represents the steps to derive this answer.
2. Ensure the question is intricate and closely related to the tool calls and the final answer.
3. Write the question from a first-person perspective, making it sound natural and human-like.

4. The question should include the necessary information about the simulation scenario and parameters in a implicit way.

The prompts using in the data construction to simulate the user's instructions:

USER_PROMPT_STEP_1:
Please call one tool related to the scenarios: {choosing_scenes}.

USER_PROMPT_STEP_2:
You can call another tool if you think the tool calls are not enough.
Or you can call the Answer_gen tool to generate the answer based on the tool calls.

USER_PROMPT_STEP_3:
It's enough. You are allowed to choose at most one another tool expect Answer_gen tool, then you must call the Answer_gen tool to generate an answer based on the tool calls.

USER_PROMPT_STEP_4:
Please generate an answer based on the tool calls.

A.3.2 Token-level Preference Sampling

The prompt using in the inference process of the Token-level Preference Sampling:

You are a tool-use professor, you can use many tools to do the following task that the user ask.

At each step, you need to analyze the status now and what to do next, with a tool call to actually execute your step.

One step just give one tool call, and you will give ONE step each time I call you.

After the call, you will get the call result, and you are now in a new state.
Then you will analyze your status now, then decide what to do next...
After many steps, you finally perform the task, then you can give your final answer.

Remember:
1. the state change is irreversible, you can't go back to one of the former state, if you want to restart the task or you want to give the final answer call the Finish tool.
2. You can do more then one trys, so if your plan is to continuously try some conditions, you can do one of the conditions per try.

Let's Begin!

Learning Rate	Warm-up Ratio	LR Scheduler	Batch Size	Epochs	LoRA rank	LoRA alpha
10^{-4}	0.1	cosine	32	5	16	32

Table 6: Hyper-parameters in experiments for training.

Query: Can you find me the closest Tesco stores near Letterkenny,Ireland please?

Apis:

```
{
  "function": [
    {
      "name": "get_tesco_locations",
      "description": "Retrieve a list of the nearest Tesco stores based on the specified location,
                     typically used for finding convenient shopping options.",
      "parameters": {
             "type": "dict",
             "required": ["location"],
             "properties": {
               "location": {
              "type": "string",
              "description": "The city and state of the user's location, in the format of 'City, State',
                             such as 'San Francisco, CA' or 'City, Country'. Use short form only for state"
               },
               "radius": {
                 "type": "integer",
                 "description": "The search radius in miles around the specified location
                                within which to find Tesco stores.",
                 "default": 10
               },
               "limit": {
                 "type": "integer",
                 "description": "The maximum number of Tesco store locations to return.",
                 "default": 5
               }
             }
      }
    },
    {
      "name": "get_news_report",
      "description": "Retrieves the latest news for a specified location formatted as 'City, State'.",
      "parameters": {
        "type": "dict",
        "required": ["location"],
        "properties": {
          "location": {
            "type": "string",
            "description": "The location for which to retrieve the news, in the format of 'City, State',
                           such as 'San Francisco, CA' or 'New York, NY'."
          }
        }
      }
    }
  ]
}
```

Golden Answer: [{"get_tesco_locations": {"location": ["Letterkenny, Ireland"], "radius": ["", 10], "limit": ["", 5]}}]

TTPA (Qwen) Answer: [{"get_tesco_locations": {"location": "Letterkenny, Ireland", "radius": 5, "limit": 3}}]

Error: ["Invalid value for parameter 'radius': 5. Expected one of ['', 10]."], "error_type": "value_error:others"

xLAM-7b-r Answer: [{"get_tesco_locations": "{"location": "Letterkenny, Ireland"}"}]

Pass!

Figure 4: The case study of BFCL. TTPA (Qwen) passes the question but is evaluated as false.