iTool: Reinforced Fine-Tuning with Dynamic Deficiency Calibration for Advanced Tool Use

Anonymous ACL submission

Abstract

Augmenting large language models (LLMs) with external tools is a promising approach to enhance their capabilities, especially for complex tasks. Synthesizing tool-use data through real-world simulations is an effective way to achieve this. However, our investigation reveals that training gains significantly decay as synthetic data increases. The model struggles to benefit from more synthetic data, and it can not equip the model with advanced tool-use capabilities in complex scenarios. Moreover, we discovered that the above limitation usually manifests as a fragment deficiency (i.e., parameter errors) in response. To this end, we propose an iterative reinforced fine-tuning strategy designed to alleviate this limitation. This strategy involves: (1) enhancing the diversity of response for synthetic data through path exploration of Monte Carlo Tree Search. (2) iteratively pinpointing the model's deficiency by constructing fine-grained preference pairs, and then improving it by preference optimization algorithms for targeted improvement. The experiments show that our method achieves 13.11% better performance than the same-size base model. It achieves an improvement of 6.5% in complex scenarios compared to the baseline, and it also outperforms larger open-source and closed-source models.

1 Introduction

Integrating LLMs with external tools significantly enhances their capability to tackle complex tasks in real-world scenarios (Li, 2025; Qu et al., 2024). For instance, the tool-use capability allows LLMs to access up-to-date information, perform precise calculations, and reduce the likelihood of hallucinations (Singh et al., 2025). This unlocks a wide range of potential applications in various domains, such as complex reasoning tasks (Li et al., 2025; Manduzio et al., 2024), and the scheduling of applications on devices (Gunter et al., 2024; Luo et al., 2025). In essence, tool use involves the following process: Given one or more tools, a user presents a question, and the LLM selects the appropriate tools from the candidate tools and performs the tool call to fulfill the user's demands. In this paper, tools are used interchangeably with APIs, functions, and plugins.

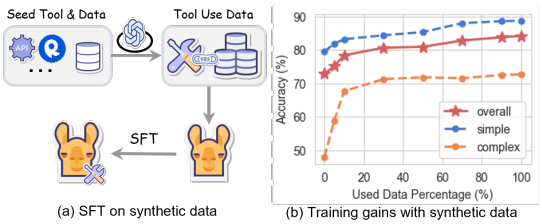

Figure 1: The training paradigm of the tool-use model under synthetic data (a). However, as shown in (b), their training gains decay significantly as the synthetic data increases, especially in complex tool-use scenarios.

Recent advancements have found that LLMs can handle simple tool use scenarios through prompt engineering (Ye et al., 2024), but they encounter difficulties with more complex real-world applications (e.g., long contexts or extensive toolsets) (Yan et al., 2024). To address this, some studies simulate real-world scenarios, such as ticketing systems, to mimic more realistic use cases (Lin et al., 2024) to collect synthetic data. Synthetic data are used in supervised fine-tuning (SFT) to improve tool use in complex scenarios, as shown in Figure 1 (a). Despite these solution strides in the development of tool-use models, our investigation reveals a critical weakness: there is a training gains decay as the synthetic tool-use data scales.

We conducted tests to explore how the performance of the model changes when synthetic data of different proportions is used, as shown in Figure 1 (b), We find that the model struggles to benefit from more synthetic data with SFT in complex scenarios. More analysis in Section 2.2 indicates

that this limitation reflects the failure of the model to extract the parameter name or infer the correct parameter value from the user query. This issue typically affects only a small fragment of the response, differing from the ground truth response.

Therefore, we attempt to alleviate the decay of training gains when using synthetic tool-use data, to enhance the ability of tool use in complex scenarios. It is not easy because it requires equipping the model with advanced contextual understanding and reasoning capabilities. Fortunately, the success of OpenA o1[1] demonstrates complex reasoning through step-by-step slow thinking (e.g., Monte Carlo Tree Search (MCTS) (Coulom, 2006)) and Reinforced Fine-Tuning (ReFT) (Luong et al., 2024) (tailors reinforcement learning and aligns with user intentions to specific tasks).

To this end, we propose a novel learning method involving (1) an MCTS-based path exploration to enhance response diversity and (2) ReFT to progressively correct the wrong fragment text of model's response. Specifically, we propose an iterative reinforced fine-tuning strategy for Tool use, named *iTool*. It first iteratively identifies complex data based on feedback from a policy model. It then performs MCTS to help explore data diversity in response, and further pinpoint wrong fragment by collecting fine-grained preference pairs from search path. Finally, a reinforcement learning policy (i.e., direct preference optimization (Rafailov et al., 2024)) is applied to align the model's response with the ground-truth response and misalign it with wrong fragment. Moreover, before iterative ReFT, we propose an easy-to-hard warm-up SFT strategy for learning from complex scenarios. Following these advancements, *iTool* demonstrates ~13% better performance than the base model. It also achieves substantial improvements in tool-use ability under complex scenarios. Despite having only 8B parameters, it outperforms larger open-source models and competes with top-tier closed-source models.

2 Problem Statement and Analysis

2.1 Task Overview

In tool use, the LLM receives a user query q along with a set of candidate tools, represented as $\mathcal{T} = \{t_0, t_1, \ldots, t_{|\mathcal{T}|}\}$. The purpose of LLM is to fulfill the user's intent by executing a specific sequence of tools. The decision process can be described as $y \sim \pi(y \mid s_0, q, \mathcal{T})$, where $\pi(\cdot)$ represents the policy model, s_0 denotes the initial task state, and y represents the actions taken by the model, such as selecting or executing a specific tool call from $\mathcal{T}$. A case is illustrated in Figure 2.

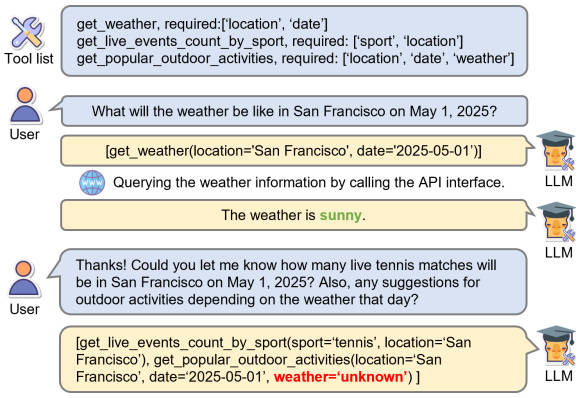

Figure 2: An illustration of tool-use. Given a user query with candidate tools, LLMs select the tool(s) from candidates, then execute the API call operation, and finally reply with a response. In the bad response, the parameter errors (i.g, red font `weather='unknown'`) account for a small fragment of the response content.

2.2 Preliminary Study

This section presents the challenges when fine-tuning models with tool-use synthetic data, and clarifies the motivation for the proposed methods.

We fine-tune the model using synthetic tool-use data of varying proportions. Specifically, training data: **ToolACE** (Liu et al., 2024) is a general tool-use dataset with up to 100K samples, and created through a novel self-evolution synthesis. Evaluation benchmark: Berkeley Function-Calling Leaderboard (**BFCL**) (Yan et al., 2024) provides a comprehensive dataset comprising 4k+ instances (updating), consisting of *Non-live* (with expert-curated simple tools), *Live* (with user-contributed complex tools), *Multi-turn* (with multi-turn & multi-step tool use) and Hallucination (i.e., relevance and irrelevance detection) samples. Here, *Non-live* denotes simple tool use scenarios (e.g., single tool), while *Live* represents more complex tool use scenarios (e.g., multiple parallel tools). For convenient understanding, in this section, we use **simple** and **complex** as aliases for the *Non-live* and *Live* metrics, respectively.

The results are depicted in Figure 1 (b). We observe that the model's performance gain declines significantly as the training data increases. Specifically, with the SFT paradigm shown in Figure 1

[1]https://openai.com/index/learning-to-reason-with-llms/

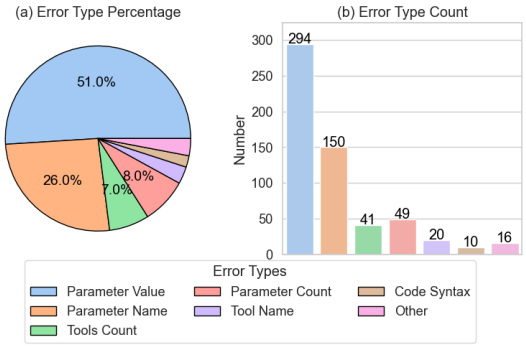

Figure 3: Error type distribution in bad cases. In bad cases, error types are highly concentrated in *Parameter Value & Name*.

(a), The model significantly enhances tool-use ability with small-scale supervised data by mimicking patterns from the training examples. However, the performance improvement significantly declines after 30% of the data is used. The model struggles to benefit from using more synthetic data, we argue that insufficient data diversity is one of the key factors.

To explore the manifestations of the above-mentioned issue, we perform a bad case analysis. We counts all error types in *Live* and *Non-live* of BFCL, and categorized the error types as shown in Figure 3. Here, *Parameter Value* error denotes the value of the parameter that does not match the ground truth. *Parameter Name* error denotes unable to identify the parameter value from the user query. For more details, see Appendix A. From Figure 3, we observed that errors are highly concentrated in *Parameter Value & Name* errors. In bad cases, parameter error constitutes a small fragment in response, while the majority remains consistent with the ground-truth. An illustration is shown in Figure 2. Therefore, trying to fix the fragment error can help alleviate the limitation of gain decay in training models.

In summary, we find that training with synthetic tool-use data causes gain decay, and the model struggles to benefit from additional such data. This limitation is reflected in the model's deficiency (i.e., parameter errors) in responses. Motivated by this line, we utilize the MCTS path to explore diversity in responses for alleviating such gains decay. We further propose an iterative ReFT strategy to progressively pinpoint and optimize the model's deficiencies.

3 Method

In this section, we provide a detailed introduction to our method. Figure 4 shows the overall architecture. It consists of warm-up training and iterative reinforcement learning.

3.1 Warm-up training

In real-world applications, the tool-use model should select multiple tools from a complex candidate toolset and schedule them correctly (a.k.a., hard mode), instead of directly using a single candidate tool to respond (a.k.a., easy mode). Similar to human learning procedures, tool learning models can benefit from an easy-to-hard curriculum during model training (Xu et al., 2020). Therefore, we propose an easy-to-hard SFT for warm-up training.

In the warm-up stage, we first divide the dataset evenly into three subsets (i.e., easy, medium, hard) based on difficulty levels. We follow the criteria: (a) the candidate toolset number; (b) the string length of the toolset; and (c) the number of tool calls needed in response to split the dataset. The specific definitions for each subset are as follows: (1) hard: `a >= 4 or b > 2000 or c >= 4`. (2) medium: `1 < a < 4 or b < 2000 or c < 4`. (3) simple: `a <= 1 and b < 1000 and c <= 1`.

$$\mathcal{D} = \mathcal{D}_{easy} \bigcup \mathcal{D}_{medium} \bigcup \mathcal{D}_{hard}. \quad (1)$$

Subsequently, we fine-tune the LLM $\mathcal{M}$ sequentially on each subset $\mathcal{D}_i$ using the supervised loss:

$$\mathcal{L}_i = -\mathbb{E}_{(q,y)\sim\mathcal{D}_i} \left[\log P_{\mathcal{M}}(y \mid q, \mathcal{T})\right], \quad (2)$$

with $\mathcal{D}_1$ (easy), $\mathcal{D}_2$ (medium) and $\mathcal{D}_3$ (hard).

The total warm-up loss is:

$$\mathcal{L}_{\text{warm-up}} = \sum_{i=1}^{N=3} \mathcal{L}_i. \quad (3)$$

3.2 MCTS-Based Iterative Reinforcement Learning

In order to alleviate training gains decreases using synthetic tool-use data for LLM, in this module, we propose an Iterative Reinforcement Learning scheme to continuously remedy this deficiency. As shown in Figure 4, it iteratively refreshes replay buffer to sample complex data and generates preference data for preference optimization.

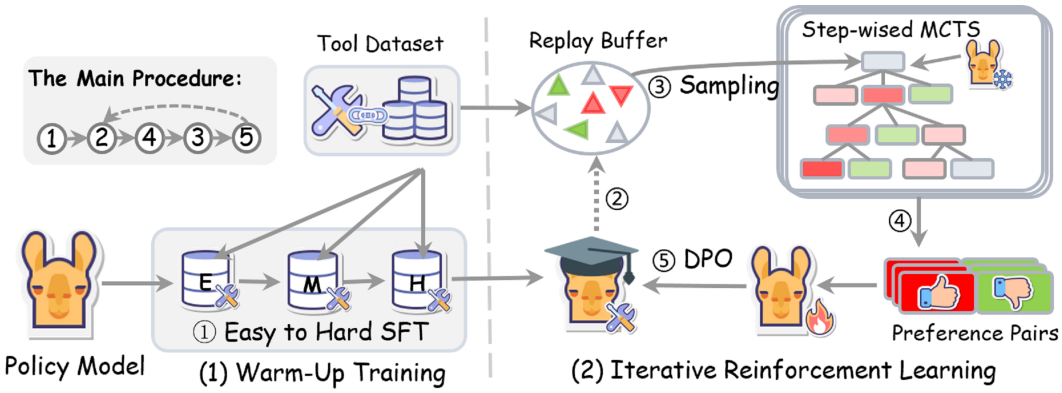

Figure 4: The overall architecture of *iTool* consists of warm-up training and iterative reinforcement learning. Specifically, after warm-up training ①, the policy model refreshes the replay buffer ② and then actively samples complex data ③. Then, step-wise MCTS ④ is performed to obtain fine-grained preference pairs for pointing out the wrong fragment in response. Finally, the models are updated via direct preference optimization ⑤ to improve response. The fire 🔥 and frozen ❄ denote parameters are updated and fixed, respectively.

Sampling complex data. Given a warm-up model from the previous stage, it is used to refresh the replay buffer by feeding back the complexity of samples. The replay buffer is initialized with a random 50% sample from the tool-use dataset. Each example in the buffer is represented as: $x_{buff} = \langle q, \mathcal{T}, c \rangle$, where c is denote the complexity of sample. In practice, model generation perplexity h is used to measure the complexity of the samples, i.e., $c = h$. The generation perplexity of the target response can be factorized as follows:

$$h = \sqrt[n]{\frac{1}{P_{\mathcal{M}}(y \mid q, \mathcal{T})}}, \tag{4}$$

where the $P_{\mathcal{M}}(y \mid q, \mathcal{T})$ is the generation probability. Since perplexity h represents the degree of generation uncertainty (Gao et al., 2024), we samples top 10% highest h data for subsequent step in each iteration.

MCTS for Step-Level Preference. The success of OpenAI o1 provides a compelling illustration of the effectiveness of step-by-step thinking. As a key algorithm, MCTS path exploration can fully traverse the search space and provide greater data diversity (Grill et al., 2020). Inspired by these, we propose to integrate MCTS into training for collecting step-level preference data.

The step-wise MCTS is achieved by breaking down the expansion step into discrete steps, transforming instance-level rewards into granular step-level signals. Specifically, it begins from a root node s_0 (i.e., user query), and unfolds in three iterative stages: selection, expansion, and backup:

(1) Select. It is guided by two key variables: $Q(s_t, a)$ is the value of taking action a in state s_t, and $N(s_t)$ is the visitation frequency of state s_t. We employ the Predictor+ Upper Confidence bounds applied to Trees (PUCT) (Rosin, 2011) to navigate the trade-off between exploring and exploiting ones. At node s_t, the subsequent node follows the formula:

$$s_{t+1} = \arg\max_{s_t} \left[Q(s_t, a) + c \cdot p(a \mid s_t) \frac{\sqrt{N(s_t)}}{1 + N(s_{t+1})} \right], \tag{5}$$

where $p(a \mid s_t) = \pi_\theta(a \mid q, \mathcal{T}, s_t)$ denotes the policy $\pi_\theta(\cdot)$'s probability distribution for generating a action step a, and c is the trade-off hyperparameter. We enforce the policy model to generate fine-grained fragments (e.g., an argument assignment operation, like `weather='unknown'` in Figure 2) by managing the termination characters (e.g., add ', .)').

(2) Expand. It occurs at a leaf node during the selection process to integrate new nodes and assess rewards. The reward $r(s_t, a)$ for executing step a in state s_t is quantified by the reward difference between states $\mathcal{R}(s_t)$ and $\mathcal{R}(s_{t+1})$, showing the benefit of action a in state s_t. As defined in Eq.6, reward computation merges outcome correctness $\mathcal{O}$ with self-evaluation $\mathcal{C}$. Following Xie et al. (2024), we define self-evaluation with Eval Prompt 8 as Eq.7.

$$\mathcal{R}(s_t) = \mathcal{O}(s_t) + \mathcal{C}(s_t), \tag{6}$$

$$\mathcal{C}(s_t) = \pi_\theta(cs \mid prompt_{eval}, q, a, \mathcal{T}, s_t), \tag{7}$$

where cs denotes the confidence score in token-level probability for correctness. Future rewards

are anticipated by simulating upcoming scenarios through roll-outs, following the selection and expansion process until reaching a terminal state (i.e., complete response or exceeds the maximum length).

(3) Backup. Once a terminal state is reached, we carry out a bottom-up update from the terminal node back to the root. We update the visit count N, the state value V, and the action value Q:

$$V(s_t) \leftarrow \sum_a N(s_{t+1}) Q(s_t, a) / \sum_a N(s_{t+1}), \quad (8)$$

$$Q(s_t, a) \leftarrow r(s_t, a) + \gamma V(s_{t+1}), \quad (9)$$

where γ is the discount for future state values.

We use the action value $\mathcal{Q}$ to indicate the preference for candidate steps, with higher values showing more preferred next steps. For each node in the search tree, we choose the steps with the highest and lowest $\mathcal{Q}$ as the preferred and dispreferred responses, respectively, and consider the prefix path as the question. See Appendix C.1 for an example.

Iterative preference optimization. Given the step-level preferences collected via MCTS, we tune the policy model via SimPO (Meng et al., 2024), a variant of DPO (Rafailov et al., 2024). Because it reduces computational overhead by eliminating the need for a reference model. After optimization, we obtain the updated policy $\pi_{\theta(i)}$ and repeat sampling the complex data process to iteratively update the policy model.

As a variant of DPO, it eliminates the need for a reference model and introduces a simple reference-free reward aligned with generation, i.e., length-normalized reward:

$$r_{\text{SimPO}}(x, y) = \frac{\beta}{|y|} \sum_{i=1}^{|y|} \log \pi_\theta(y_i \mid x, y_{<i}), \quad (10)$$

where β is a constant that controls the scaling of the reward difference. Using the shorthand $h^{y_w}_{\pi_\theta} = \frac{\beta}{|y_w|} \log \pi_\theta(y_w|x)$, $h^{y_l}_{\pi_\theta} = \frac{\beta}{|y_l|} \log \pi_\theta(y_l|x)$, at the i-th iteration, given a batch of preference data $\mathcal{D}_i$ sampled with the latest policy $\pi_{\theta(i-1)}$, we denote the policy objective $\ell_i(\theta)$ as follows:

$$\ell_i(\pi_\theta) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}_i} \left[\log \sigma \left(h^{y_w}_{\pi_\theta} - h^{y_l}_{\pi_\theta} - \gamma \right) \right], \quad (11)$$

where $\gamma > 0$ represents the target reward margin, ensuring that the preferred response's reward exceeds that of the dispreferred one; y_w and y_l represent the step-level preferred and dispreferred responses, respectively.

4 Experiments

4.1 Experimental Setup

We take the widely used open-source LLM, LLaMA3.1-8B-Instruct as our base model. We use synthetic data from ToolACE for experiments, 90% for warm-up training, and 50% for reinforcement learning to balance performance and cost. For warm-up training, we adopt the parameter-efficient training strategy LoRA (Hu et al., 2022). For reinforcement learning, we employ SimPO, a variant of DPO, for preference optimization, utilizing the QLora parameter-efficient training strategy (Dettmers et al., 2024). For more implementation details and preferences optimization analysis, see Appendix B.

Evaluation Dataset. In addition to **BFCL**, we use **API-Bank** (Li et al., 2023), which consists of 314 tool-use dialogues and 753 API calls. This dataset evaluates models' abilities to correctly invoke a known API (L-1) based on a query and to retrieve and call APIs from a tool list (L-2).

Baselines We compare the overall performance with the state-of-the-art closed-source models (e.g., GPT-series, Gemini and open-source models (e.g., Llama-3.1-8B-Instruct, Qwen2.5-7B (Team, 2024)), as well as fine-tuned open-source models with tool-use dataset, including ToolACE-8B (fine-tuning Llama-3.1-8B-Instruct on ToolACE) model, xLAM-series (Zhang et al., 2024) and Hammer-series (Lin et al., 2024).

4.2 Overall Performance

The overall performance of *iTool-8B* and baseline models are shown in Table 1 and Table 2. It shows our model consistently achieves corresponding a better performance at comparable scales ($\sim$ 8B). Specifically, it shows consistent advantageous performance on API-Bank and BFCL compared with open-source models, and also outperforms most closed-source and larger open-source models in BFCL (e.g., GPT-4-series models). For example, it outperforms xLAM-8x22b-r by 5.27 in the overall accuracy metrics. Moreover, it demonstrates its superiority in challenging scenarios (e.g., *Live*), which indicates our method learn advanced tool-use capabilities effectively from synthetic data. This is primarily due to our iterative ReFT strategy, which continuously pinpoints and optimizes the model's deficiencies.

Rank	Overall Acc	Model	Non-live	Live	Multi turn	Rel / Irrel
1	63.26	♣ **iTool-8B (FC)**	88.82	**78.29**	23.84	84.90/80.72
2	**62.19**	♠ GPT-4o-2024-08-06 (FC)	86.15	75.43	**25.00**	63.41/82.93
3	61.89	♠ GPT-4-turbo-2024-04-09 (FC)	88.80	76.23	24.88	73.17/79.76
4	60.47	♠ GPT-4o-mini-2024-07-18 (FC)	83.72	70.19	27.50	80.49/71.77
5	60.44	♣ ToolACE-8B (FC)	88.94	74.99	17.38	80.49/85.71
6	58.15	♠ GPT-4o-mini-2024-07-18 (Prompt)	88.69	74.63	11.13	75.61/81.00
7	57.99	♣ xLAM-8x22b-r (FC)	87.51	71.97	14.50	85.37/67.29
8	57.92	♠ Gemini-1.5-Flash-002 (Prompt)	87.60	76.28	9.88	85.37/78.54
9	57.69	♣ Hammer2.0-7b (FC)	88.54	69.79	14.75	95.12/68.46
10	57.45	♠ o1-mini-2024-09-12 (Prompt)	83.84	75.39	13.12	48.78/88.04
11	56.80	♡ mistral-large-2407 (FC)	81.41	68.37	20.62	75.61/49.44
12	56.51	♠ Gemini-1.5-Pro-002 (Prompt)	**89.63**	74.41	5.50	65.85/77.30
13	55.86	♠ Gemini-1.5-Flash-001 (Prompt)	85.74	69.21	12.62	82.93/67.84
14	55.78	♠ GPT-4-turbo-2024-04-09 (Prompt)	88.80	69.04	9.50	82.93/58.95
15	55.10	♠ Gemini-1.5-Pro-001 (Prompt)	86.17	73.12	6.00	56.10/85.00
16	54.41	♣ xLAM-7b-r (FC)	80.86	67.88	14.50	97.56/64.05
17	54.27	♡ Qwen2.5-7B-Instruct (Prompt)	85.58	65.97	11.25	92.68/64.95
18	53.67	♡ Llama-3.1-70B-Instruct (Prompt)	87.50	61.13	12.38	92.68/58.38
19	53.66	♡ Gemma-2-27b-it (Prompt)	87.39	69.48	4.12	87.80/68.76
20	53.00	♠ GPT-3.5-Turbo-0125 (FC)	78.52	61.22	19.25	97.56/35.16
21	52.50	♡ Gemma-2-9b-it (Prompt)	84.52	69.21	3.75	87.80/72.45
22	51.59	♣ Hammer2.0-1.5b (FC)	84.44	63.22	7.13	92.68/60.64
23	51.50	♡ Meta-Llama-3-70B-Instruct (Prompt)	85.10	66.15	3.25	92.68/52.78
27	50.15	♡ Llama-3.1-8B-Instruct (Prompt)	81.15	57.93	11.38	78.05/41.62
28	49.02	♣ xLAM-8x7b-r (FC)	73.93	69.12	4.00	87.80/68.12
29	48.82	♡ Qwen2.5-1.5B-Instruct (Prompt)	53.99	61.71	6.62	75.61/67.17
42	42.98	♡ Llama-3.2-3B-Instruct (Prompt)	11.11	50.91	4.00	63.41/68.81

Table 1: The leaderboard of different models in four tool-use scenarios of BFCL (v3) benchmark . The top 20 models and baselines are listed for comparison. FC denotes the model is tailored for functional calling. Rel and Irrel denote relevance and irrelevance detection, respectively, indicating whether to call a tool or not. ♠ denotes closed-source model, ♡ denotes open-source base model, ♣ denotes open-source fine-tuned model.

Model	API-Bank L1	API-Bank L2
♠ GPT-3.5-turbo-0125	70.43	**52.59**
♠ GPT-4-0613	75.94	48.89
♠ GPT-4-turbo-2024-04-09	72.43	39.26
♠ GPT-4o-mini-2024-07-18	74.69	45.93
♠ **GPT-4o-2024-05-13**	**76.19**	42.96
♡ Alpaca-7B	24.06	5.19
♡ ChatGLM-6B	23.62	13.33
♣ Lynx-7B	49.87	30.37
♣ xLAM-7b-fc-r	32.83	21.48
♡ LLaMA-3.1-8B-Instruct	71.18	37.04
♡ Qwen2.5-7B-Instruct	72.83	41.98
♣ ToolACE-8B	75.94	47.41
♣ **iTool-8B**	**78.89**	**52.87**

Table 2: Accuracy performance comparison on API-Bank evaluation system. **Bold** values represent the highest performance.

Models	Non-live	Live	Multi-turn
Base Model	81.15	57.93	11.38
+ *SFT*	88.94 ↑9.6	74.99 ↑17	17.38 ↑6.0
+ *warm-up*	88.35 ↓0.6	75.84 ↑1.0	19.65 ↑2.3
+ *IRT (iTool)*	88.82 ↑0.5	78.29 ↑3.2	23.84 ↑4.2
Total	↑9.5	↑21.2	↑12.5

Table 3: The module ablation performance (↑ = increase, ↓ = decrease, values are relative percentage changes from the previous row).

4.3 Ablation Analysis

4.3.1 Module Ablation

To evaluate the effectiveness of the two components in our method, we conduct an ablation study in: (1) the warm-up training phase (*w/o warm-up*). (2) the Iterative Reinforcement Learning (IRT) module (*w/o IRT*). We adopt LLaMA-3.1-8B-Instruct as the *Base* model for benchmarking, ensuring a consistent baseline across all experimental conditions. From Figure 3, we find that all components are essential within our method. The model achieves a comparable level to *SFT* on the Non-live metric, but each module brings substantial improvements on the complex-scenario metrics (Live and Multi). Specifically, the warm-up training and IRT modules individually contribute improvements of 2.3 and 4.2 points, respectively, on the Multi-turn metric. Cumulatively, it gets a 6.5 improvement over *SFT* and a 12.5 gain relative to *Base*, highlighting effects in complex, multi-step reasoning tasks.

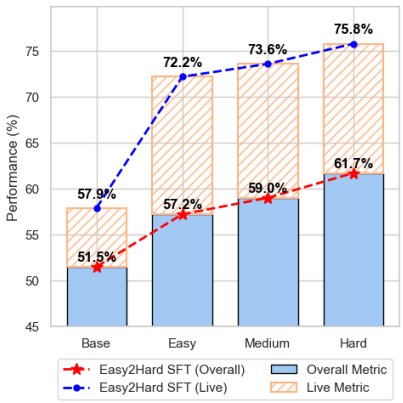

Figure 5: The performance progression of easy to hard warm-up training on *Live* and *Overall* metrics.

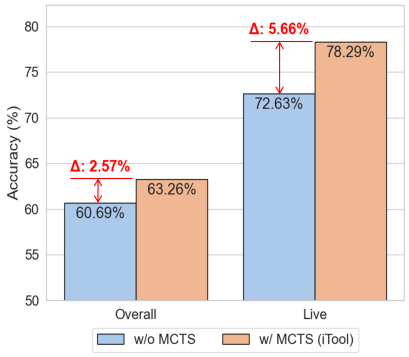

Figure 6: The result of ablation study on MCTS in *iTool* on key metrics.

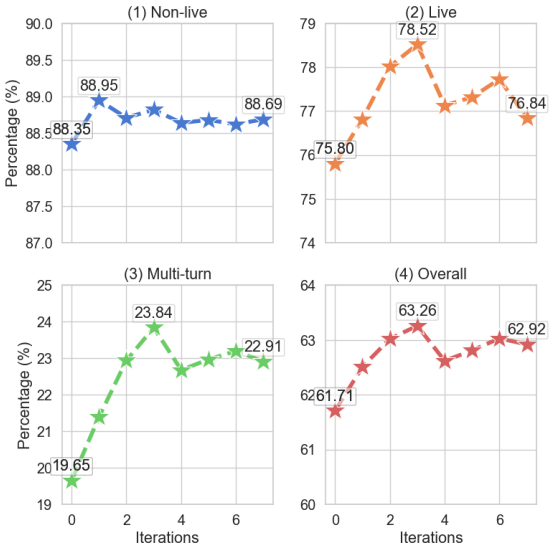

Figure 7: The performance variation of our model with the increase of iterations.

4.3.2 Deeper Ablation

(1) In warm-up training, we conducted a study on the easy2hard SFT strategy. We present the performance progression from easy to hard and compare it with base model. The experimental results are summarized in Figure 5. From the results, we observe that our strategy shows a gradual improvement. There is a significant leap from *base* to *easy*, and the second largest improvement occurs from the *medium* to *hard*. In the synthetic data, the model can quickly learn the task patterns of tool use from the easier stages, which in turn benefits the harder scenario. This indicates that the model benefits from the curriculum learning process that goes from easy to hard.

(2) In iterative reinforcement learning, we conducted a study on MCTS and iteration counts. The results are illustrated in Figure 6 and 7 respectively. To replace MCTS, we sample four responses from the policy model and select the responses with the highest and lowest probabilities as preference pairs. These pairs are then used for subsequent preference optimization (w/o MCTS). From Figure 6, we observe that the model's performance deteriorates when MCTS is replaced. From Figure 7, we observe that as iterations increase, our method initially shows an upward trend before declining. The model performs best around 3 iterations, especially in the Multi-turn and Live scenarios. This indicates that MCTS can effectively mitigate the issue of insufficient data diversity with a small number of iterations. However, excessive iterations can lead to overfitting, resulting in a decrease in data diversity.

4.3.3 Base Model Analysis.

To further validate the effectiveness of base models, we applied our method to other base models. Due to computational resource constraints, we compared the following base models ($< 10B$): (1) Llama-3.2-3B-Instruct, (2) Qwen2.5-7B-Instruct (Team, 2024). From Table 4, our method exhibits remarkably stable performance across different base models. This highlights the robustness of our method in various base models. On Llama-3.2-3B, our method improved performance by 18% over the base model. On Qwen2.5-7B, it achieved the best performance at 63.22%.

4.4 Training Gains Analysis

To analyze the training gains of our method, as detailed in Section 2.2, we test the training gains of our method. From Figure 8, our method shows greater training gains as the data scale increases in *Live* and *Overall*. Unlike SFT, whose training benefit curve flattens beyond 30%, our model exhibits a steeper curve in the Live metric. This suggests that

Base Model	Method	Oveall	Non-live	Live	Multi-turn	Rel / Irrel
Llama-3.1-8B-Instruct	Vanilla	50.15	81.15	57.93	11.38	78.05 / 41.62
	Baseline	60.44	88.94	74.99	17.38	80.49 / 85.71
	Our	63.26	88.82	78.29	23.84	84.90 / 80.72
Llama-3.2-3B-Instruct	Vanilla	42.98	11.11	50.91	4.00	63.41 / 68.81
	Baseline	58.22	89.27	73.90	11.50	84.37 / 78.20
	Our	62.93	90.59	76.43	15.82	84.27 / 87.82
Qwen2.5-7B-Instruct	Vanilla	54.27	85.58	65.97	11.25	92.68 / 64.95
	Baseline	60.69	90.02	76.23	15.92	73.47 / 86.98
	Our	63.93	91.29	82.28	22.38	80.28 / 85.12

Table 4: The accuracy performance comparison of base models with different methods on BFCL benchmark. *Vanilla* denotes source base model, *Baseline* denotes supervised fine-tuned base model, *Our* denotes *iTool*.

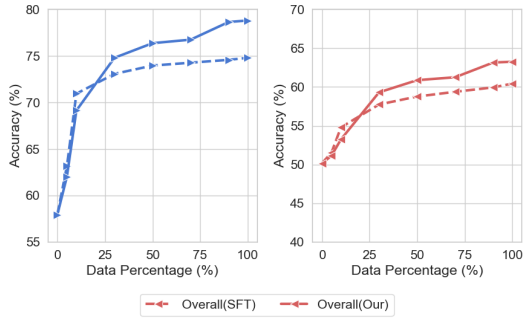

Figure 8: The change curve of training gains as the data scale increases on key metrics.

our model can alleviate the internal decay of training gains by enhancing its advanced capabilities in complex scenarios.

5 Related Work

5.1 Tool use of LLMs

Pioneering works like Toolformer (Schick et al., 2023) and ToolAlpaca (Tang et al., 2023) have explored the potential of LLMs in tool use. Previously, several tuning-free methods were proposed, which involves manipulating prompts (e.g., (Xu et al., 2023; Shi et al., 2024; Qiao et al., 2024)) or enhancing execution frameworks (e.g., ReAct (Yao et al., 2023), RestGPT (Song et al., 2023)) to unlock inherent capabilities.

Due to the limitation of user-defined tools in prompts of the above methods, tuning-based methods with synthetic data have been focused. ToolLlama (Qin et al., 2023) notably expanded the toolset and investigated the impact of data scaling on performance. More efficient data synthesis techniques have been proposed for tool use (e.g., ToolACE (Liu et al., 2024), BUTTON (Chen et al., 2024), and xLAM (Zhang et al., 2024)).

5.2 Reinforcement Learning

Learning from human feedback is crucial in aligning LLMs with human intentions (Leike et al., 2018), which is known as reinforcement learning. ReFT enhances this process by combining reinforcement learning with SFT to optimize model performance using reward signals. Online reinforcement learning algorithms (Schulman et al., 2017; Zheng et al., 2023) are complex and difficult to optimize. Recently, Direct Preference Optimization (DPO) (Rafailov et al., 2024), a simpler offline algorithm, reparameterizes the reward function to learn a policy model from preference data directly, enhancing simplicity and training stability. Besides, a variety of preference optimization objectives have been proposed, e.g., SimPo (Meng et al., 2024), IPO (Azar et al., 2024), ORPO (Hong et al., 2024) and KTO (Ethayarajh et al., 2024).

Further studies have extended this approach to an iterative training setup, by continuously updating the reference model with the most recent policy model or generating new preference pairs at each iteration (Dong et al., 2024; Yuan et al., 2024; Kim et al., 2024; Xiong et al., 2024)

6 Conclusion

Equipping LLMs with external tools is becoming a viable method to enhance their capabilities. In this paper, we study enhancing the advanced tool-use capabilities in a complex scenario from synthetic data. We find that there are training decay issues when training with synthesized tool-use data. To alleviate it, we propose an iterative reinforced fine-tuning strategy. It can continually pinpoint the model's wrong fragments in its responses and address these deficiencies by preference optimization. The experimental results demonstrate the effectiveness of the proposed method.

7 Limitaiton

While our study has achieved notable advancements, it is important to acknowledge several limitations that could be addressed in future work. First, the iterative reinforcement learning process (particularly the Monte Carlo Tree Search) requires substantial computational resources to generate fine-grained preference data. Although it is difficult to solve, we have effectively implemented parameter constraints to manage computational costs efficiently (e.g., 7 hours on 8 V100 GPUs per iteration), achieving a balance between computational feasibility and model performance. Additionally, due to limited computing resources, we are not able to validate our method on larger 30B or 70B base models. Finally, when analyzing the synthetic tool-use data, only a single dataset was tested. Testing more publicly available datasets would strengthen the validity and persuasiveness of the conclusions. We will address these limitations in our future work.

References

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR.

Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. 2024. Facilitating multi-turn function calling for llms via compositional instruction tuning. *arXiv preprint arXiv:2410.12952*.

Rémi Coulom. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. 2024. Rlhf workflow: From reward modeling to online rlhf. *arXiv preprint arXiv:2405.07863*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

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

Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. 2020. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pages 3769–3778. PMLR.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, et al. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11170–11189.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Dahyun Kim, Yungi Kim, Wonho Song, Hyeonwoo Kim, Yunsu Kim, Sanghoon Kim, and Chanjun Park. 2024. sdpo: Don't use your data all at once. *arXiv preprint arXiv:2403.19270*.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. 2018. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116.

Wenjun Li, Dexun Li, Kuicai Dong, Cong Zhang, Hao Zhang, Weiwen Liu, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025. Adaptive tool use in large language models with meta-cognition trigger. *arXiv preprint arXiv:2502.12961*.

Xinzhe Li. 2025. A review of prominent paradigms for llm-based agents: Tool use, planning (including rag), and feedback learning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9760–9779.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu

Zhou, Cheng Cheng, Yin Zhao, et al. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

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

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. *Preprint*, arXiv:2401.08967.

Graziano A Manduzio, Federico A Galatolo, Mario GCA Cimino, Enzo Pasquale Scilingo, and Lorenzo Cominelli. 2024. Improving small-scale large language models function calling for reasoning tasks. *arXiv preprint arXiv:2410.18890*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Huajun Chen, et al. 2024. Autoact: Automatic agent learning from scratch for qa via self-planning. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Christopher D Rosin. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230.

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

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

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10642–10657, Miami, Florida, USA. Association for Computational Linguistics.

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

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

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

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

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, Chi Jin, Tong Zhang, and Tianqi Liu. 2024. Building math agents with multi-turn iterative preference learning. *Preprint*, arXiv:2409.02392.

Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. 2020. Curriculum learning for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6095–6104.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-sourced large language models. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard.

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*.

Junjie Ye, Yilong Wu, Sixian Li, Yuming Yang, Tao Gui, Qi Zhang, Xuanjing Huang, Peng Wang, Zhongchao Shi, Jianping Fan, et al. 2024. Tl-training: A task-feature-based framework for training large language models in tool use. *arXiv preprint arXiv:2412.15495*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. 2024. xlam: A family of large action models to empower ai agent systems. *CoRR*.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, et al. 2023. Secrets of rlhf in large language models part i: Ppo. *arXiv preprint arXiv:2307.04964*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

A Details in Preliminary Study

A.1 Descriptions of error types

Here is the descriptions of all error types.

- **Parameter Value**. The value or type of the parameter does not match the ground truth.
- **Parameter Name**. Unable to identify the parameter value from the user query.
- **Parameter Count**. Incorrect number of parameters; required parameters are missing.
- **Tools Count**. The wrong number of tools was called.
- **Tool Name**. There was an error when calling the tool name, such as calling a non-existent tool name or a tool name that does not match the ground truth.
- **Code Syntax**. The tool call does not comply with the syntax of Python, Java, or JavaScript.
- **Other**. Errors other than those mentioned above.

B Complementary Experiments

B.1 More Implementation Details

The experiments were conducted using the publicly available training repository, LLaMA-Factory (Zheng et al., 2024). The training of our model can be done within 28 hours with 8 NVIDIA Tesla V100-SXM2-32GB GPUs. For the training model, we take the best performance checkpoint on the valid dataset.

The Implementation Settings. Due to resource constraints, we employ a parameter-efficient training strategy using LoRA (with `rank=16` and `alpha=32`) during the SFT warm-up phase, and QLoRA (a quantization method from the bitsandbytes [2] library with 4 bits) during the reinforcement learning (RL) phase. We utilize a cosine learning rate scheduler with a warm-up ratio of 0.1. More detailed training settings are shown in Table 5.

Stage	epoch	lr	batch size
SFT	3	easy: 5e-5 medium: 2e-5 hard: 1e-5	64
RL	2	1e-6	64

Table 5: The detailed training settings in our method. `lr` denotes learning rate. `batch size` denotes the total batch size, equals 1 (per device) times 8 (accumulation steps) times 8 (devices).

Implementation Settings in MCTS-base RL. In *Expand* phase of MCTS, the prompt for self-evaluation is shown in Table 8. When calculating the confidence score for correctness, we evaluate the token-level probabilities of a policy model across four options (`A, B, C, D`) with respective weights of 1.0, 0.1, -1.0, and -2.0. We sample the model's responses four times and use the weighted average of these samples as the final confidence score.

To ensure the quality of the sampled preference data, we exclude the following data: (1) pairs with candidate step similarity above 95%, (2) pairs with a Q-value difference less than 0.1, and (3) accepted samples with a Q-value below 0.3. In MCTS, to control algorithm overhead, we limit the following parameters: (1) `depth`, the maximum depth of the search tree, (2) `width`, the maximum number of

[2] https://github.com/TimDettmers/bitsandbytes

child nodes per node, (3) `simulation`, the maximum number of simulation steps in *Expand* phase, and (4) `iterations`, the maximum number of iterations to construct the MCTS search tree. We summarize these parameters in Table 6.

Parameters	Value	Parameters	Value
`depth`	3	c	1.0
`width`	3	`temperature`	1.5
`simulation`	2	`seed`	42
`iterations`	5		

Table 6: The parameters setting in MCTS. c denotes the degree of exploration in the *Select* phase.

B.2 Preference Algorithm Analysis

In iterative reinforcement learning, we also explore different preference optimization algorithms. Besides the widely used DPO (Rafailov et al., 2024), we also explored SimPO (Meng et al., 2024), IPO (Azar et al., 2024), and ORPO (Hong et al., 2024). DPO reparameterizes the reward function to learn a policy model from preference data directly. IPO is a theoretically grounded approach method that avoids DPO's assumption that pairwise preferences can be replaced with pointwise rewards. ORPO introduces a reference-model-free odd ratio term to directly contrast winning and losing responses with the policy model and jointly trains with the SFT objective. SimPO aligns the reference-free reward function in the preference optimization objective with the generation metric. For fair comparisons, we start these algorithms from the same SFT checkpoints, the reference model is initialized as the policy model.

For these algorithms, we conducted a thorough search for the optimal hyperparameter settings to ensure a fair comparison. The results of hyperparameter settings are shown in Table 7. The results of different preference optimization algorithm with optimal hyperparameter settings are shown in Figure 9. From the result, we find *iTool* with SimDPO achieved the best performance. Different preference algorithms do not create significant performance gaps except for ORPO.

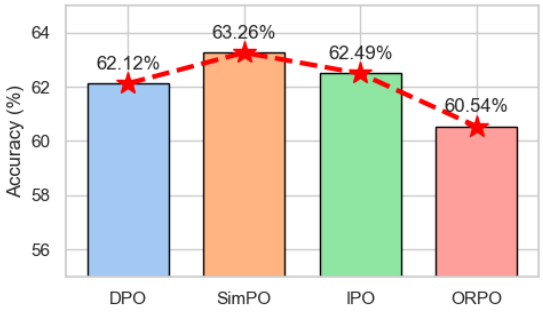

Figure 9: The performance *iTool* using different preference optimization algorithms on BFCL.

C Case Analysis

C.1 An Example of Preference Pair

Table 9 illustrates a preference pair example. The `chosen` response correctly employs the "Get Trending Result" tool with suitable parameters for the user's request. Conversely, the `rejected` response is improperly formatted, omits necessary parentheses, and incorrectly assigns the value 1 to the `timeframe` parameter, showcasing an erroneous application of the tool.

Table 10 presents another case of preference pair, sampled during the MCTS research tree as depicted in Figure 10. In this scenario, the user's query lacks the specific details necessary for the functions mentioned (i.e., `reviews` for 'reviewAnalytics.extractSentiment' and `metrics` for 'socialTrends.fetchTrendingProducts'). The assistant's chosen response correctly identifies the need for these parameter values, whereas the rejected response incorrectly hallucinates when recognizing these parameters.

Method	Objective	Hyperparameters	Best Setting
DPO	$-\log\sigma\left(\beta\log\frac{\pi_\theta(y_w\|x)}{\pi_{\text{ref}}(y_w\|x)}-\beta\log\frac{\pi_\theta(y_l\|x)}{\pi_{\text{ref}}(y_l\|x)}\right)$	$\beta\in[0.01,0.05,0.1]$ $lr\in[1e-6,5e-7,3e-7]$	$\beta=0.1$ $lr=3e-7$
IPO	$\left(\log\frac{\pi_\theta(y_w\|x)}{\pi_{\text{ref}}(y_w\|x)}-\log\frac{\pi_\theta(y_l\|x)}{\pi_{\text{ref}}(y_l\|x)}-\frac{1}{2\tau}\right)^2$	$\tau\in[0.01,0.05,0.1]$ $lr\in[1e-6,5e-7,3e-7]$	$\tau=0.1$ $lr=1e-6$
ORPO	$-\log p_\theta(y_w\|x)-\lambda\log\sigma\left(\log\frac{p_\theta(y_w\|x)}{1-p_\theta(y_w\|x)}-\log\frac{p_\theta(y_l\|x)}{1-p_\theta(y_l\|x)}\right)$, where $p_\theta(y\|x)=\exp\left(\frac{1}{\|y\|}\log\pi_\theta(y\|x)\right)$	$\lambda\in[0.01,0.05,0.1]$ $lr\in[1e-6,5e-7,3e-7]$	$\lambda=0.1$ $lr=3e-7$
SimPO	$-\log\sigma\left(\frac{\beta}{\|y_w\|}\log\pi_\theta(y_w\|x)-\frac{\beta}{\|y_l\|}\log\pi_\theta(y_l\|x)-\gamma\right)$	$\beta\in[2.0,2.5]$ $\gamma\in[0.5,1.0,1.4]$ $lr\in[1e-6,5e-7,3e-7]$	$\beta=2.5$ $\gamma=0.5$ $lr=1e-6$

Table 7: The search for optimal hyperparameter settings of different preference optimization algorithms.

Prompt 1: Eval Prompt

Ground Truth Response: `{gt_ans}`
Generated Response by Model: `{response}`

User Instruction:
Please assess the quality of the generated response relative to the ground truth response.
Note: A generated response that is a fragment of the ground truth response is also excellent.

Evaluation Criteria:
1. Function Name: Is the name of all the function called correct?
2. Parameter Count: Is the number of parameters for all the function correct?
3. Parameter Names: Are the names of all the parameters for the function correct?
4. Parameter Value/Types: Are the value/types of all the parameters for the function correct?
5. Semantic Similarity: Is the generated response semantically close to the ground truth response?

Please directly choose from the following options to judge the overall quality:
(A) Excellent: The generated response meets all criteria and is almost identical to the ground truth response.
(B) Acceptable: The generated response meets most criteria but has minor discrepancies.
(C) Fair: The generated response meets some criteria but has significant issues.
(D) Poor: The generated response fails to meet most or all criteria.

ASSISTANT: The option of overall quality is
You are an AI specialized in tool use.
Your task is to assess the potential veracity of `{placeholder}`.

Table 8: The Eval Prompt for self-evaluation in Eq. 7 of Section 3.2.

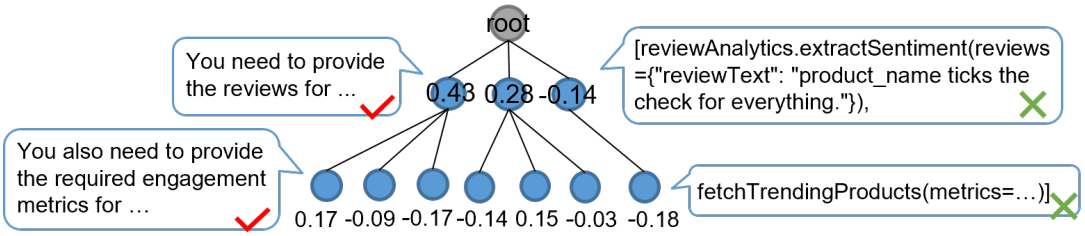

Figure 10: The illustration of example 2 in Table 10 for preference pair derived from MCTS. The floating-point values of nodes denote the Q-value in MCTS.

Example 1	

Tool list	[{
	"name": "Get Trending Result",
	"description": "Retrieves trending search results from Google Trends for the United States.",
	"parameters": {
	"type": "dict",
	"properties": {
	"category": {
	"description": "The category of trending results (e.g., news, entertainment, etc.)",
	"type": "string"
	},
	"timeframe": {
	"description": "The timeframe for which to retrieve trending results (e.g., today, this week, etc.)",
	"type": "string"
	}
	},
	"required": ["category"]
	},
	"required": null
	},{
	"name": "Get Server Time",
	"description": "Returns the current server time in a standardized format",
	"parameters": {
	"type": "dict",
	"properties": {},
	"required": []
	},
	"required": null
	}]

User	Can you tell me the latest trending topics in entertainment today?
Assistant	
chosen	[Get Trending Result(category="entertainment", timeframe="today")]
rejected	[Get Trending Result category="entertainment", timeframe="1"]

Table 9: The example 1 of preference pair derived from MCTS.

Example 2	

Tool list	[{
	"name": "fetchTrendingProducts",
	"description": "Fetches trending products based on reviews and engagement metrics.",
	"parameters": {
	"metrics": {
	"type": "array",
	"items": {
	"productID": "string",
	"engagementScore": {
	"likes": "int",
	"shares": "int"
	} } } }
	},{
	"name": "extractSentiment",
	"description": "Extracts sentiment scores from reviews.",
	"parameters": {
	"reviews": {
	"type": "array",
	"items": {
	"reviewText": "string",
	"reviewDate": {
	"type": "string",
	"pattern": "^\\w+ \\d{1,2}, \\d{4}$"
	} } } }
	}]

User	I'm running a marketing campaign and need sentiment analysis from this month's reviews of our main product. We had 300 likes and 50 shares yesterday. Please analyze trends and sentiment for our competitor's product too; they had 450 likes and 75 shares.
Assistant	
chosen	You need to provide the reviews for the sentiment analysis of the flagship product and the competitor's product.
rejected	[reviewAnalytics.extractSentiment(reviews={"reviewText": "product_name ticks the check for everything."}),

Table 10: The example 2 of preference pair derived from MCTS.