ToolACE-R: Model-aware Iterative Training and Adaptive Refinement for Tool Learning

Xingshan Zeng[1], Weiwen Liu[2*], Xu Huang[3], Zezhong Wang[1], Lingzhi Wang[4], Liangyou Li[1], Yasheng Wang[1], Lifeng Shang[1], Xin Jiang[1], Ruiming Tang[1], Qun Liu[1]

[1]Huawei Technologies Co., Ltd
[2]Shanghai Jiao Tong University
[3]University of Science and Technology of China
[4]Harbin Institute of Technology, Shenzhen
zeng.xingshan@huawei.com,wwliu@sjtu.edu.cn

Abstract

Tool learning, which allows Large Language Models (LLMs) to leverage external tools for solving complex user tasks, has emerged as a promising avenue for extending model capabilities. However, existing approaches primarily focus on data synthesis for fine-tuning LLMs to invoke tools effectively, largely ignoring how to fully stimulate the potential of the model. In this paper, we propose ToolACE-R, a novel framework that includes both model-aware iterative training and adaptive refinement for tool learning. ToolACE-R features a model-aware iterative training procedure that progressively adjust training samples based on the model's evolving capabilities to maximize its potential. Additionally, it incorporates self-refinement training corpus which emphasizes LLM's ability to iteratively refine their tool calls, optimizing performance without requiring external feedback. Furthermore, we introduce adaptive self-refinement for efficient test-time scaling, where the trained model can autonomously determine when to stop the process based on iterative self-refinement. We conduct extensive experiments across several benchmark datasets, showing that ToolACE-R achieves competitive performance compared to advanced LLMs. The performance can be further improved efficiently through adaptive self-refinement. These results highlight the effectiveness and generalizability of ToolACE-R, offering a promising direction for more efficient and scalable tool learning.

Introduction

Tool learning, enabling Large Language Models (LLMs) to leverage external tools to address complex user requirements, has gained increasing attention. With tool integration, LLMs can access up-to-date information, perform intricate computations, and utilize third-party services, significantly expanding their capabilities beyond simple natural language communication with humans (Qu et al. 2024b). While tool invocation requires the LLMs to demonstrate strong understanding, reasoning, and instruction-following skills, customized fine-tuning is currently the dominant approach for enabling models to call external tools (Liu et al. 2024b,c; Patil et al. 2023; Qin et al. 2023).

Due to the limited availability of high-quality data, existing research has primarily focused on developing effective and efficient methods for data synthesis using advanced models (Abdelaziz et al. 2024; Liu et al. 2024b,c; Wang et al. 2024b; Prabhakar et al. 2025). However, data synthesized by advanced models from different sources can lead to compatibility issues. Specifically, when synthesized samples exceed the model's current knowledge, they may undermine the model's performance or lead to hallucinations (Kang et al. 2024; Ren et al. 2024). As a result, determining which training data samples are *appropriate* for a given model remains an unresolved challenge.

Additionally, the potential for *maximizing a model's intrinsic capabilities* remains underexplored in the field of tool learning. On one hand, prior work has seldom investigated how to fully exploit existing training data through techniques such as data augmentation. On the other hand, scaling test-time computation – despite its demonstrated effectiveness in enhancing LLM reasoning (Brown et al. 2024; Snell et al. 2024) – has received limited attention in tool learning scenarios. A promising approach involves iteratively refining the model's outputs and selecting the majority answer (Snell et al. 2024). However, user queries vary widely in complexity, ranging from simple to highly intricate. Scaling test-time computation indiscriminately is inefficient, as models can often correctly answer straightforward questions without the need for additional refinement. Therefore, an adaptive strategy for scaling test-time computation is needed to dynamically determine the appropriate level of computational effort required for each query.

To address the challenges outlined above, we propose ToolACE-R, a novel pipeline for stimulating model potential in tool learning, encompassing both training and inference procedures. Specifically, we propose a model-aware iterative training framework that enables LLMs to learn tool invocation in alignment with their evolving capabilities. This is achieved through a novel model-aware difficulty metric, guiding the training process in an iterative manner. Additionally, a simple yet effective self-refinement data incorporation strategy is employed as a form of data augmentation. Furthermore, by preserving refinement samples that in-

*Corresponding Author.

clude identical cases, LLMs are encouraged to learn appropriate stopping criteria during the iterative process. Building upon this training procedure, we integrate adaptive self-refinement into iterative inference, allowing models to autonomously determine when to halt refinement, thereby improving inference-time efficiency.

We have conducted extensive experiments on several representative tool-calling benchmarks, such as the Berkeley Function Call Leaderboard (BFCL) (Yan et al. 2024) and API-Bank (Li et al. 2023), to assess the effectiveness and efficiency of ToolACE-R. Experimental results demonstrate that ToolACE-R achieves competitive performance compared to advanced API-based models such as GPT-4o, and can be further enhanced through adaptive self-refinement. Additional analyses highlight the contributions of the proposed modules and examine the generalizability of our method across different model backbones and sizes.

The contributions of this work are summarized as follows:

- We propose a model-aware iterative training procedure, augmented with self-refinement corpus construction, to maximize the potential of LLMs for tool learning.
- Building upon the training process, we integrate adaptive self-refinement into iterative inference, enabling models to autonomously decide when to halt the refinement process and allowing for more efficient use of computational resources when scaling test-time compute.
- We conduct extensive experiments on several representative tool calling benchmarks, revealing effectiveness and efficiency of our method.

Related Work

Tool Learning. Tool learning methods generally fall into two categories, tuning-free and tuning-based approaches. Tuning-free methods employ prompting techniques without additional training (Mialon et al. 2023; Hsieh et al. 2023; Ruan et al. 2023). A notable example method is ReAct (Yao et al. 2023), which can alternate between reasoning and action for complex tasks. Recent work (Qu et al. 2024a) improves tool understanding by iteratively refining documentation. Tuning-based methods, which directly enhance tool calling ability via specialized fine-tuning, have recently gained much more attention (Qin et al. 2023; Schick et al. 2023; Patil et al. 2023; Tang et al. 2023; Abdelaziz et al. 2024). A large amount of them focus on data synthesis for improved generation procedures (Liu et al. 2024b,c; Wang et al. 2024b), often overlooking the model's inherent potential. Zeng et al. (2025) also apply iterative training but based on DPO (Rafailov et al. 2023) which introduces more complexity. While some others integrate tool feedback for refinement (Du, Wei, and Zhang 2024; Wang et al. 2024a), our approach is more streamlined, directly refining outputs in an iterative manner without external or any textual feedback to maximize model efficiency.

Data Selection. Selecting high-quality training samples is essential for fine-tuning LLMs (Albalak et al. 2024). A small set of high-quality data can effectively harness the model's potential, rather than relying on large quantities of data (Zhou et al. 2023; Liu et al. 2023). While earlier works primarily emphasize general data quality aspects, such as diversity and complexity, recent studies advocate for model-specific data selection (Du, Zong, and Zhang 2023; Li et al. 2024). This kind of approach is based on the observation that data distributions that significantly deviate from the base model's are challenging for the model to learn from and may even degrade the performance (Ren et al. 2024). Our method aligns with this perspective, defining a new metric to identify and select suitable training samples based on the model to be trained.

Self Refinement. Previous research has demonstrated that LLMs can refine their own generations through either self-feedback (Madaan et al. 2023; Weng et al. 2023) or external feedback (Qu et al. 2024c; Xu et al. 2024). Nevertheless, it remains challenging for LLMs to assess the correctness of their refined output autonomously (Huang et al. 2024), where scaling test-time compute with iterative refinement still often depends on post-processing techniques, such as majority voting (Snell et al. 2024). In this work, we propose a simple yet effective method that enables LLMs to adaptively self-refine their outputs, improving scaling efficiency.

Methodology

Problem Formulation

Given a user query q and a set of candidate tools $T = \{t_1, t_2, \ldots, t_n\}$, the objective of tool learning is to generate the correct tool invocations. This includes selecting the most appropriate tools and extracting the suitable parameter values. Specifically, the goal is to produce:

$$A = [t_1(a_1), \ldots, t_m(a_m)] = f_\theta(\langle q, T \rangle) \tag{1}$$

where t_j and a_j represent the j-th invoked tool and its corresponding argument, respectively, with $1 \leq j \leq m$ and m being the total number of tool invocations needed. The function $f_\theta(\cdot)$ denotes the generation process of the LLM with parameters θ. For each a_j, it might include several parameters and the corresponding values, denoted as $a_j = [p_1 : v_1, ..., p_i : v_i, ...]$, where p_i is the parameter name, and v_i is the corresponding value.

The training samples for tool learning typically leverage q and T as context, and A as output (the ground-truth tool invocation), denoted as $\{\langle q, T \rangle, A\}$.

ToolACE-R is trained with a model-aware iterative training pipeline and can iteratively self-refine during inference in an adaptive manner. Figure 1 shows the overall procedure.

Model-Aware Iterative Training

We begin model training with an instruction-tuned base model to ensure it possesses fundamental instruction-following capabilities required by our model-aware module. For the training process, we first collect a set of off-the-shelf samples and augment them via self-refinement. Rather than using these samples directly, we apply a selection criterion grounded in our defined notion of model-aware difficulty to curate the data. This curated dataset is then used to train the model and is iteratively re-curated based on new trained model throughout the training process.

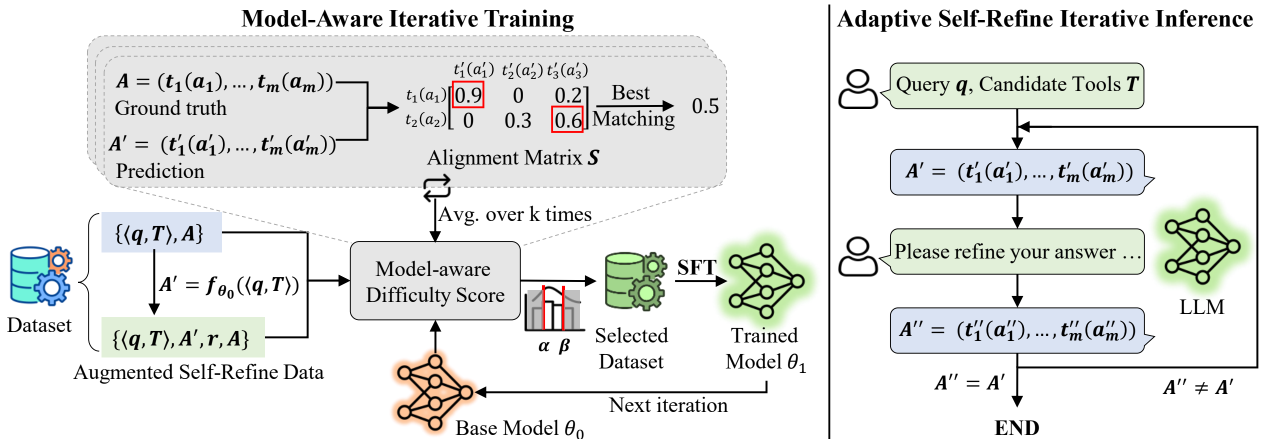

Figure 1: The overview of ToolACE-R, including two components, for training and inference, respectively.

Model-Aware Difficulty. We introduce a *model-aware difficulty* metric that quantifies the inherent challenge a specific training sample poses to a given model. Rather than relying solely on static heuristics or dataset-level statistics, our definition measures how well the model can approximate the desired tool invocation behavior before training, thereby capturing a performance-aware notion of difficulty.

Let a training sample be denoted as $\{\langle q, T\rangle, A\}$, where $A = [t_1(a_1), \dots, t_m(a_m)]$ is the reference tool invocation sequence. Given a base model θ_0, we obtain the model's prediction as $A' = [t'_1(a'_1), \dots, t'_{m'}(a'_{m'})] = f_{\theta_0}(\langle q, T\rangle)$. To assess the similarity between the predicted and reference sequences, we adopt Jaccard-style overlap to define a *parameter-level alignment score* between individual tool calls $t_i(a_i)$ and $t'_j(a'_j)$ as follows:

$$S_{ij} = \begin{cases} \frac{|a_i \cap a'_j|}{|a_i \cup a'_j|} & \text{if } t_i = t'_j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Here, $|a_i \cap a'_j|$ denotes the number of matching arguments between a_i and a'_j, i.e., parameters with both identical names and values. $|a_i \cup a'_j|$ represents the total count of distinct arguments across both calls. Parameters with the same name but differing values are counted separately to penalize incorrect assignments.

We aggregate all pairwise scores into a similarity matrix $S \in \mathbb{R}^{m \times m'}$. To compute the best structural alignment between A and A', we frame the task as a bipartite matching problem over the index sets $\mathcal{I} = 1, \dots, m$ and $\mathcal{J} = 1, \dots, m'$, and apply the Hungarian algorithm (Kuhn 1955) to identify the optimal matching $\mathcal{M}$ that maximizes the total alignment score:

$$\mathcal{M} = \arg\max_{\mathcal{M} \in \text{Matchings}(\mathcal{I}, \mathcal{J})} \sum_{(i,j) \in \mathcal{M}} S_{ij} \tag{3}$$

where Matchings$(\mathcal{I}, \mathcal{J})$ denotes the set of all one-to-one matchings between $\mathcal{I}$ and $\mathcal{J}$, i.e., subsets of $\mathcal{I} \times \mathcal{J}$ in which each element in $\mathcal{I}$ and $\mathcal{J}$ appears in at most one pair. Using this optimal alignment, we define the *overlap score* between A and A' as:

$$O(A, A') = \frac{\sum_{(i,j) \in \mathcal{M}} S_{ij}}{\max(m, m')} \tag{4}$$

This score reflects the degree to which the model reproduces the reference tool call, normalized by the longer of the two sequences to account for missing or superfluous tool calls.

To incorporate model uncertainty and account for variance in its outputs, we evaluate the model over k independent attempts, resulting in outputs $A'^{(1)}, \dots, A'^{(k)}$. The *model-aware difficulty* of a sample is then defined as:

$$D(\langle q, T\rangle, A) = 1 - \frac{1}{k} \sum_{l=1}^{k} O(A, A'^{(l)}) \tag{5}$$

This difficulty score $D \in [0, 1]$ quantifies the average divergence between the model's output and the reference across multiple attempts. A lower score indicates that the model consistently produces accurate or nearly accurate tool calls, while a higher score suggests that the model struggles with the sample, failing to replicate even partial correctness. For instance, $D = 0$ implies perfect performance across all outputs, whereas $D = 1$ reflects complete failure in reproducing any reference calls.

This model-aware difficulty serves as a valuable signal for iterative training, enabling more effective model development in tool-use scenarios.

Data Augmentation with Self-Refinement. To augment the training corpus and establish the model's self-refinement capability, we additionally construct self-refinement data based on the original training samples. Specifically, we create a self-refinement sample as a multi-turn interaction, where the first turn is the initial response from the model, and the second turn includes a refinement prompt from the user followed by the refined answer (typically the ground-truth tool invocation A): $\{\langle q, T\rangle, A_1, r, A\}$. Here, the model to be trained will first produce its output tool invocation A_1. We then concatenate a refinement requirement prompt r (a generalized sentence like "Please refine your answer...") and the final ground-truth answer A.

Notably, there may be samples where $A_1 = A$, meaning the refined answer is identical to the previous answer. We retain these samples to enable the model to learn that ”no further changes are needed when the answer is perfect.” This is a crucial aspect of establishing the model's adaptive, iterative self-refinement, and we will provide further details in later section.

Each self-refinement sample can also be estimated its difficulty using Eq. 5, by replacing $\{\langle q, T\rangle\}$ with $\{\langle q, T\rangle, A_1, r\}$ as context. In this way, all the training samples including the constructed self-refinement samples can be estimated and labeled with a difficulty score.

Model-Aware Iterative Data Selection and Training. We hypothesize that the most suitable training samples are those within the model's current capability, either too difficult or too simple samples are ineffective for model training. To that end, we empirically set two thresholds α and β, where $\alpha < \beta$. For any training sample $\{C, A\}$ (where C indicates the context, $\langle q, T\rangle$ or $\{\langle q, T\rangle, A_1, r\}$ for self-refinement data), if its estimated difficulty score $\alpha < D(C, A) < \beta$, we retain it as an effective training sample and add to the actual training set $\mathcal{D}_0$.

Given the selected training set $\mathcal{D}_0$, we apply supervised fine-tuning to the base model θ_0 for one epoch, producing θ_1. This constitutes the first iteration. In subsequent iterations, we treat the trained model as the new base model, reconstruct the self-refinement samples, re-evaluate the difficulty, and re-select samples with the same criterion for fine-tuning the updated model.

Ideally, with each iteration, the model's performance improves, leading to an update of data difficulty and so new training samples might be included matching model's current capability. The iteration will continue until the performance improvement is marginal, indicating that the model has saturated on this set of training samples. The resulting model will then be our final ToolACE-R model.

Adaptive Self-Refine Iterative Inference

With model-aware iterative training, our ToolACE-R model can directly generate tool invocations based on a user query and candidate tools, and refine its answer when necessary.

We propose an adaptive self-refine procedure that enables ToolACE-R to iteratively refine its own output and autonomously determine when to halt the iteration. Details are shown in Figure 1 (right hand side). Specifically, given a user query q and candidate tools T, ToolACE-R first generates the tool invocation A'. In each subsequent iteration, ToolACE-R leverages the answer from the previous iteration as context and refines it. As mentioned earlier, ToolACE-R has learned that when the answer is sufficiently accurate, it can produce the same answer in subsequent refinement. Therefore, the iteration continues until the answers from two consecutive iterations are identical. We refer to this procedure as adaptive self-refinement, where the model itself determines when to stop the iterations. To prevent infinite iteration, we set a maximum iteration limit n. Through adaptive self-refinement, ToolACE-R is able to improve its performance, particularly when facing difficult cases.

Experiments

Experimental Settings

Datasets and Models. We use a subset of ToolACE (Liu et al. 2024b) training data as our training data, where we only retain the samples with single-turn tool calling for simplicity. We select LLaMA3.1-8B-Instruct (AI@Meta 2024) as base model in our main experiments. We also conduct experiments on Qwen2.5-Instruct-series (Yang et al. 2024) (0.5B, 1.5B, 3B and 7B) and Mistral-7B-Instruct-v0.2 (Jiang et al. 2023), to validate the generalizability of ToolACE-R. We compare with the state-of-the-art API-based models, including GPT-series and Gemini-2.5-Pro, as well as open-source models like DeepSeek-V3 (Liu et al. 2024a), Llama3.1-8B/70B-Instruct (AI@Meta 2024) and Qwen2.5-7B-Instruct (Yang et al. 2024). We also compare with fine-tuned tool calling models like ToolACE-8B (Liu et al. 2024b), Hammer2.1-7B (Lin et al. 2024) and xLAM-2-8b-fc-r (Zhang et al. 2024; Prabhakar et al. 2025).

Benchmarks and Evaluation. We conduct experiments on several representative benchmarks, including the Berkeley Function Call Leaderboard (BFCL) (Yan et al. 2024), ACEBench (Chen et al. 2025), API-Bank (Li et al. 2023) and ToolAlpaca (Tang et al. 2023). To minimize external interference and simplifying data preparation, we mainly evaluate performance on single-turn tool calling queries. For BFCL, we evaluate the Non-live and Live subsets, which correspond to synthetic test cases and real-world scenarios, respectively. Each subset includes four categories: Simple, Multiple, Parallel, and Multiple Parallel. For ACEBench, we evaluate only the English normal category, excluding multi-turn cases. This includes the test samples in atom category and single-turn category. For API-Bank and ToolAlpaca, we focus on the first step tool calling samples, disregarding further tool or retrieval feedback. All performance is reported in terms of accuracy, where only cases with entirely correct tool calling are counted as correct.

Implementation Details. We employ parameter-efficient fine-tuning method LoRA (Hu et al. 2022) given resource constraints. All model modules are configured for LoRA fine-tuning, with a rank of 16 and an alpha value of 32. Training is performed with a global batch size of 64 and a learning rate of 1×10^{-4}, following a cosine learning rate schedule with a warmup ratio of 0.1. For difficulty estimation, we set the generation temperature to 1.0 to encourage diversity and allow the model to explore a broader range of outputs. Model generates 8 times (i.e. $k = 8$) for each sample to ensure robustness. During data selection, we set $\alpha = 0$ and $\beta = 0.9$, which means that we discard samples that are extremely easy (difficulty score equals to 0, i.e., the model always answer fully correctly) or too difficult (most attempts are total incorrect). For the constructed refinement data, we apply a loss mask to the first assistant turn of the conversation, as it may involve an incorrect tool invocation, making it unsuitable for the model to learn from this turn. During evaluation, we adopt greedy search to ensure stability. Unless otherwise specified, , the maximum iteration time for adaptive self-refinement is set to $n = 5$.

Models	Non-Live				Live				Overall		
	Simple	*Multiple*	*Parallel*	*Multiple Parallel*	*Simple*	*Multiple*	*Parallel*	*Multiple Parallel*	*Non-live*	*Live*	*Overall*
GPT-4o-2024-11-20	77.17	95.00	93.50	85.00	<u>84.50</u>	79.30	<u>87.50</u>	70.83	87.67	80.24	83.96
GPT-4.1-2025-04-14	**80.50**	94.00	93.00	87.50	**85.66**	76.54	**93.75**	75.00	88.75	78.46	83.60
GPT-4o-mini-2024-07-18	<u>80.08</u>	90.50	89.50	87.00	81.40	76.73	**93.75**	<u>79.17</u>	86.77	77.87	82.32
Gemini-2.5-Pro-Exp-03-25	67.50	91.00	83.50	73.00	79.46	64.96	81.25	<u>79.17</u>	78.75	68.17	73.46
DeepSeek-V3	78.67	95.50	91.00	**91.50**	83.72	**82.15**	81.25	62.50	89.17	**82.09**	<u>85.63</u>
Llama3.1-70B-Inst	77.92	<u>96.00</u>	<u>94.50</u>	**91.50**	78.29	76.16	<u>87.50</u>	66.67	<u>89.98</u>	76.53	83.26
Llama3.1-8B-Inst	71.50	93.50	86.50	86.00	73.26	68.95	56.25	50.00	84.37	69.28	76.83
Qwen2.5-7B-Inst	70.50	94.00	90.50	82.50	74.81	71.89	62.50	62.50	84.38	72.17	78.28
ToolACE-8B (FC)	76.67	93.50	90.50	89.50	73.26	76.73	81.25	70.83	87.54	76.02	81.78
Hammer2.1-7B (FC)	78.08	95.00	93.50	88.00	76.74	77.40	81.25	70.83	88.65	77.20	82.92
xLAM-2-8b-fc-r (FC)	73.08	93.50	87.00	84.00	74.81	66.29	56.25	50.00	84.40	67.51	75.95
ToolACE-R (FC)	79.75	**97.00**	**95.50**	**91.50**	82.95	<u>81.10</u>	**93.75**	**87.50**	**90.94**	<u>81.72</u>	**86.33**

Table 1: Accuracy comparison on BFCL (Last updated on 2025-04-25). The best results in each category are marked in **bold**. The second best results are <u>underlined</u>. FC indicates the models are fine-tuned for function calling.

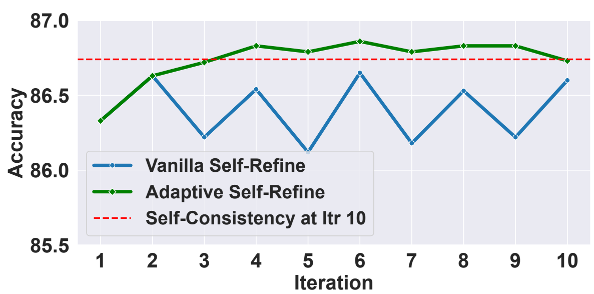

Figure 2: Performance of different iterative self-refine methods. "Vanilla Self-Refine" always picks the final answer at each iteration. "Adaptive Self-Refine" is our proposed method, while "Self-Consistency" selects the majority answer during the iteration.

Models	ACEBench	APIBank	ToolAlpaca
GPT-4o	**87.00**	<u>77.16</u>	83.87
GPT-4o-mini	81.63	73.10	83.87
Llama3.1-8B-Inst	50.63	60.41	79.03
ToolACE-R	83.88	75.89	<u>85.48</u>
+ Adaptive SR	<u>84.00</u>	**81.22**	**88.71**

Table 2: Accuracy comparison on ACEBench, API-Bank and ToolAlpaca. "SR" is short for "Self-Refine".

Main Results

Results on BFCL. Table 1 presents the comparison results on BFCL, including detailed results for each category. The following key observations can be made:

- The performance gap between API-based and open-source models is minimal. Large open-source models like DeepSeek-V3 show comparable or even superior overall performance to GPT models and Gemini-2.5-Pro.
- Specialized fine-tuned models significantly benefit from domain-specific fine-tuning, where smaller models (7/8B) are allowed to compete with larger general models like Llama3.1-70B-Inst.
- Our model, ToolACE-R, which leverages our proposed model-aware iterative training procedure, achieves the best overall performance, surpassing both large API-based and open-source models.

To further assess the effectiveness of our adaptive self-refine method for iterative inference, we conducted experiments comparing with two other inference-time scaling approaches. The first, termed "Vanilla Self-Refine," always selects the last answer at each iteration as the final answer. The second method, "Self-Consistency at Itr n", chooses the majority answer from the answers collected during iterations 1 to n. The results of these inference methods on BFCL, are shown in Figure 2.

As observed, the performance of "Vanilla Self-Refine" fluctuates across iterations, indicating that the model does not always refine the answer correctly. In contrast, our "Adaptive Self-Refine" method demonstrates more stability across iterations, suggesting that adjacent identical answers serve as a reliable signal for stopping further iterations. Our statistics also show that the average iteration time of our method on BFCL is 2.4 (when maximum iteration time $n = 5$), indicating that most cases terminate early thus achieving consistent performance. Our method achieves competitive performance compared to the strong baseline "Self-Consistency at Itr 10", further validating advantages.

Results on More Benchmarks. To provide a more comprehensive evaluation, we continue to conduct experiments on three other representative benchmarks ACEBench, API-Bank, and ToolAlpaca, focusing on single-turn tool-calling scenarios. The results are summarized in Table 2.

On ACEBench, ToolACE-R demonstrates an impressive absolute improvement of over 30% compared to the base model, Llama3.1-8B-Inst, highlighting the effectiveness of

Models	Non-live	Live	Overall
ToolACE-R + Adaptive SR	**91.32**	**82.34**	**86.83**
ToolACE-R	90.94	81.72	86.33
- Data Selection	88.96	81.13	85.04
- SR Data	88.75	80.09	84.42
Llama3.1-8B-Inst (base)	84.37	69.28	76.83

Table 3: Ablation Study of our proposed modules on BFCL. "SR" is short for "Self-Refine".

our training pipeline. However, the benefit of Adaptive Self-Refine is marginal, likely due to the presence of particularly challenging cases where the model struggles to correctly refine its answers.

For both API-Bank and ToolAlpaca, ToolACE-R achieves substantial improvements. Moreover, when combined with Adaptive Self-Refine, the performance sees further gains, even surpassing GPT-4o. These results validate the potential of Adaptive Self-Refine to progressively refine tool invocation answers based solely on the model's own capabilities, opening up new possibilities for scaling inference in tool learning scenarios.

Ablation Study. To evaluate the contribution of each proposed module, we conduct an ablation study by progressively removing them. The corresponding results on the BFCL dataset are presented in Table 3. As shown, the additional inference cost introduced by adaptive self-refinement leads to performance gains in both categories. Further removing model-aware data selection and the constructed self-refinement data both result in a clear performance drop, with the absence of model-aware data selection causing the most significant degradation. These findings demonstrate the effectiveness of our proposed modules, especially for the model-aware part which mainly comes from the novel definition of data sample difficulty.

Effects of Model-Aware Iterative Training

In this subsection, we explore several different choices when applying model-aware iterative training, to validate the effectiveness of our current proposal.

Choices of Selection Thresholds. In our main experiments, we set $\alpha = 0$ and $\beta = 0.9$ to discard samples that are either extremely simple or too difficult for model training. Here we experiment with more different choices of the two thresholds. We turn to another three kinds of choices, preserving simpler samples, preserving more difficult samples, or preserving samples in more medium difficulty. Table 4 displays the evaluation results on BFCL. As shown, setting $\alpha = -1$ and $\beta = 0.9$, which includes even extremely simple cases (i.e., $\alpha = 0$), leads to a slight performance drop while increasing training time due to the inclusion of a larger training corpus. In contrast, preserving only relatively difficult cases (by setting $\alpha = 0.1$ and $\beta = 2$) results in a notable performance degradation across both categories, with an overall decrease of approximately 1%. This suggests that samples beyond the model's current capability may hinder effective learning and limit performance improvements based on its existing potential. We further evaluate a stricter setting ($\alpha = 0.1, \beta = 0.8$), which discards more simple and difficult cases. This also results in performance drop, especially for Live category. The likely reason is that an excessive number of training samples are discarded (reducing the dataset to approximately half the size of our main setting) while retaining only a small portion of the harder examples.

Thresholds	Non-live	Live	Overall
$\alpha = 0, \beta = 0.9$ **(ToolACE-R)**	90.94	81.72	86.33
$\alpha = -1, \beta = 0.9$	90.73	81.27	86.00
$\alpha = 0.1, \beta = 2$	89.27	81.35	85.31
$\alpha = 0.1, \beta = 0.8$	90.83	80.38	85.61

Table 4: Performance on BFCL when applying different thresholds during model-aware data selection. We always select samples that their difficulty score satisfy $\alpha < D < \beta$.

Models	Non-live	Live	Overall
$k = 8$ **(ToolACE-R)**	90.94	81.72	86.33
+ Adaptive SR	91.32	82.34	86.83
$k = 1$	90.94	80.98	85.96
+ Adaptive SR	90.71	82.01	86.36

Table 5: Performance on BFCL when setting k value in Eq. 5 as 8 and 1. "SR" is short for "Self-Refine".

Choices of k. In Eq. 5, we define the difficulty score by allowing the model to make k attempts, thereby better exploring its potential and producing a more robust estimate. To evaluate the impact of different k values, we compare our main setting ($k = 8$) with a reduced setting ($k = 1$). The corresponding results are shown in Table 5. The model trained with $k = 1$ exhibits a notable performance drop, particularly in the Live category, where difficult cases play a more critical role. Upon examining the final training set, we observe that approximately 10% of the samples within the difficulty range $[0.8, 0.9]$ under the $k = 8$ setting are excluded when using $k = 1$. This provides evidence that a larger k value yields a more stable difficulty estimation and facilitates the selection of more appropriate training samples.

Choices of Models. We further examine the "model-aware" part, where we use the base model to be trained to evaluate the difficulty of training samples. We conduct an additional experiment. Instead of using data selected by the base model Llama3.1-8B-Inst, we train Llama3.1-8B-Inst with data selected by Qwen2.5-3B-Inst and Qwen2.5-7B-Inst, as well as with the intersection of all three models' selected data – i.e., only the samples that all three models perform similar (within the same difficulty scope) are included in the dataset. The results of the trained models on BFCL, along with the amounts of corresponding training data (in the first iteration), are presented in Table 6. As shown, performance drops when using data selected by other models, even when the data volume is larger. The intersection dataset, which contains less data, achieves comparable

Models	Live	Overall	Amount
ToolACE-R	81.72	86.33	120K
Select w/ Qwen2.5-3B	80.83	85.50	125K
Select w/ Qwen2.5-7B	81.27	85.97	134K
Intersection Data	81.37	86.08	108K

Table 6: Performance on BFCL when using different selected data to train Llama3.1-8B-Inst, along with the data amount. "Qwen2.5-3/7B" is short for Qwen2.5-3/7B-Inst.

Models	Non-live	Live	Overall
Qwen2.5-7B-Inst	84.38	72.17	78.28
+ Original Data	88.29	78.39	83.34
+ ToolACE-R	89.23	81.20	85.21
Mistral-7B-Inst-v0.2	55.25	55.07	55.16
+ Original Data	84.06	70.17	77.12
+ ToolACE-R	87.98	80.01	84.00

Table 7: Performance of using different models on BFCL.

results. This observation underscores the benefits of model-aware data selection.

Iterative Training. To examine the training dynamics, we visualize the iterative training process by illustrating the trends in both accuracy and the amount of training data across iterations, as shown in Figure 3. Iteration 0 corresponds to the base model, LLaMA3.1-8B-Inst. As depicted, the number of selected training samples decreases monotonically with each iteration, primarily because more samples are assigned a difficulty score of 0 and consequently excluded from the training corpus. Meanwhile, accuracy improves rapidly during the first two iterations, peaking at iterations 2 and 3. Beyond this point, performance plateaus, with no further gains observed in subsequent iterations. This pattern suggests that excessive training may lead to overfitting on the filtered dataset. Therefore, to save the training cost, we recommend iterative training at most 3 times.

Further analysis

In this subsection, we examine the generalizability of our method by varying the base model and exploring the scaling effects across different model sizes.

Different Backbones. To assess the generalizability of ToolACE-R, we conduct experiments with different base models, including Qwen2.5-7B-Inst and Mistral-7B-Inst-v0.2. The results, along with the original performance and performance after training with the full original training set, are presented in Table 7. As shown, fine-tuning with the full training set leads to a significant performance boost for both models, particularly for the relatively weaker Mistral-7B-Inst-v0.2, which sees an improvement of over 20%. Moreover, our method further enhances performance, using exactly the same original data, demonstrating its effectiveness.

Model Size Scaling. We further examine whether our method can be applied to models of varying sizes and whether performance improves as model size increases. We use Qwen2.5-Instruct models of different sizes, including 0.5B, 1.5B, 3B, and 7B, as base models. The results, shown in Figure 4, display both the original performance and the performance after applying ToolACE-R. Applying ToolACE-R consistently improves performance across all sizes, particularly for smaller models like 0.5B and 1.5B. This demonstrates the effectiveness of ToolACE-R across a range of model sizes.

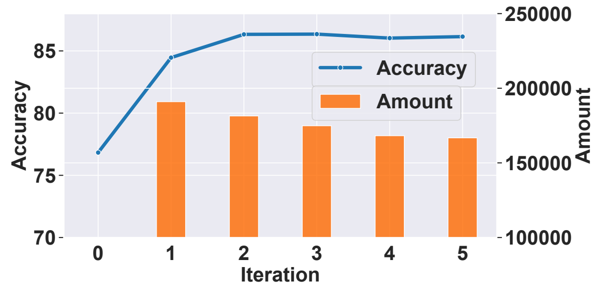

Figure 3: Performance and training amount of each iteration during training. Iteration 0 means the model before training.

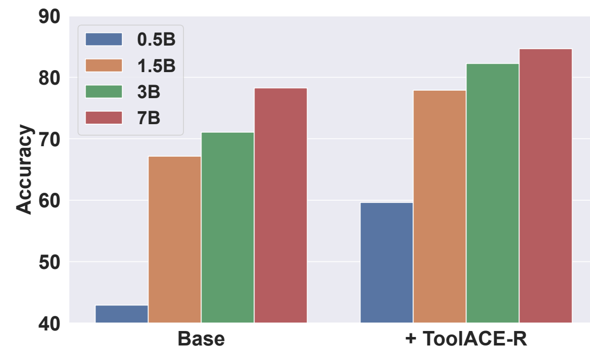

Figure 4: Performance of Qwen2.5-Series. "Base" refers to the Qwen2.5-Instuct models without any training, while "+ ToolACE-R" uses our method.

Conclusion

In this paper, we introduce ToolACE-R, a tool learning technique designed to enable LLMs to learn to effectively leverage external tools. ToolACE-R consists of two components: model-aware iterative training and adaptive self-refine iterative inference. Model-aware iterative training maximizes the model's potential by iteratively selecting the most appropriate samples for training, with a novel definition of sample difficulty. Adaptive self-refinement further enhances performance by allowing the model to autonomously determine when to stop refining its own answers during iterative inference. Experiments demonstrate that ToolACE-R consistently improves performance across various benchmarks and is effective with different backbone models. Additionally, our analyses highlight the impact of the proposed modules on model performance.

Acknowledgments

The research described in this paper is partially supported by The National Natural Science Foundation of China (Grant No. 62502118, 62502310).

References

Abdelaziz, I.; Basu, K.; Agarwal, M.; Kumaravel, S.; Stallone, M.; Panda, R.; Rizk, Y.; Bhargav, G.; Crouse, M.; Gunasekara, C.; et al. 2024. Granite-Function Calling Model: Introducing Function Calling Abilities via Multi-task Learning of Granular Tasks. *arXiv preprint arXiv:2407.00121*.

AI@Meta. 2024. Llama 3 Model Card.

Albalak, A.; Elazar, Y.; Xie, S. M.; Longpre, S.; Lambert, N.; Wang, X.; Muennighoff, N.; Hou, B.; Pan, L.; Jeong, H.; Raffel, C.; Chang, S.; Hashimoto, T.; and Wang, W. Y. 2024. A Survey on Data Selection for Language Models. ArXiv:2402.16827 [cs].

Brown, B.; Juravsky, J.; Ehrlich, R.; Clark, R.; Le, Q. V.; Ré, C.; and Mirhoseini, A. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Chen, C.; Hao, X.; Liu, W.; Huang, X.; Zeng, X.; Yu, S.; Li, D.; Wang, S.; Gan, W.; Huang, Y.; et al. 2025. ACEBench: Who Wins the Match Point in Tool Learning? *arXiv preprint arXiv:2501.12851*.

Du, Q.; Zong, C.; and Zhang, J. 2023. Mods: Model-oriented data selection for instruction tuning. *arXiv preprint arXiv:2311.15653*.

Du, Y.; Wei, F.; and Zhang, H. 2024. AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls. ArXiv:2402.04253 [cs].

Hsieh, C.-Y.; Chen, S.-A.; Li, C.-L.; Fujii, Y.; Ratner, A.; Lee, C.-Y.; Krishna, R.; and Pfister, T. 2023. Tool Documentation Enables Zero-Shot Tool-Usage with Large Language Models. arXiv:2308.00675.

Hu, E. J.; yelong shen; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Huang, J.; Chen, X.; Mishra, S.; Zheng, H. S.; Yu, A. W.; Song, X.; and Zhou, D. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations*.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Kang, K.; Wallace, E.; Tomlin, C.; Kumar, A.; and Levine, S. 2024. Unfamiliar finetuning examples control how language models hallucinate. *arXiv preprint arXiv:2403.05612*.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2): 83–97.

Li, M.; Zhang, Y.; Li, Z.; Chen, J.; Chen, L.; Cheng, N.; Wang, J.; Zhou, T.; and Xiao, J. 2024. From Quantity to Quality: Boosting LLM Performance with Self-Guided Data Selection for Instruction Tuning. In Duh, K.; Gomez, H.; and Bethard, S., eds., *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 7602–7635. Mexico City, Mexico: Association for Computational Linguistics.

Li, M.; Zhao, Y.; Yu, B.; Song, F.; Li, H.; Yu, H.; Li, Z.; Huang, F.; and Li, Y. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. ArXiv:2304.08244 [cs].

Lin, Q.; Wen, M.; Peng, Q.; Nie, G.; Liao, J.; Wang, J.; Mo, X.; Zhou, J.; Cheng, C.; Zhao, Y.; et al. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.

Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liu, W.; Huang, X.; Zeng, X.; Hao, X.; Yu, S.; Li, D.; Wang, S.; Gan, W.; Liu, Z.; Yu, Y.; Wang, Z.; Wang, Y.; Ning, W.; Hou, Y.; Wang, B.; Wu, C.; Wang, X.; Liu, Y.; Wang, Y.; Tang, D.; Tu, D.; Shang, L.; Jiang, X.; Tang, R.; Lian, D.; Liu, Q.; and Chen, E. 2024b. ToolACE: Winning the Points of LLM Function Calling. arXiv:2409.00920.

Liu, W.; Zeng, W.; He, K.; Jiang, Y.; and He, J. 2023. What Makes Good Data for Alignment? A Comprehensive Study of Automatic Data Selection in Instruction Tuning. ArXiv:2312.15685 [cs].

Liu, Z.; Hoang, T.; Zhang, J.; Zhu, M.; Lan, T.; Kokane, S.; Tan, J.; Yao, W.; Liu, Z.; Feng, Y.; et al. 2024c. APIGen: Automated Pipeline for Generating Verifiable and Diverse Function-Calling Datasets. *arXiv preprint arXiv:2406.18518*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. ArXiv:2303.17651 [cs].

Mialon, G.; Dessì, R.; Lomeli, M.; Nalmpantis, C.; Pasunuru, R.; Raileanu, R.; Rozière, B.; Schick, T.; Dwivedi-Yu, J.; Celikyilmaz, A.; Grave, E.; LeCun, Y.; and Scialom, T. 2023. Augmented Language Models: a Survey. arXiv:2302.07842.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Prabhakar, A.; Liu, Z.; Yao, W.; Zhang, J.; Zhu, M.; Wang, S.; Liu, Z.; Awalgaonkar, T.; Chen, H.; Hoang, T.; et al. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; and Wen, J.-r. 2024a. FROM EXPLORATION TO MASTERY: ENABLING LLMS TO MASTER TOOLS VIA SELF-DRIVEN INTERACTIONS.

Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; and Wen, J.-R. 2024b. Tool Learning with Large Language Models: A Survey. ArXiv:2405.17935 [cs].

Qu, Y.; Zhang, T.; Garg, N.; and Kumar, A. 2024c. Recursive Introspection: Teaching Language Model Agents How to Self-Improve. ArXiv:2407.18219 [cs].

Rafailov, R.; Sharma, A.; Mitchell, E.; Ermon, S.; Manning, C. D.; and Finn, C. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. ArXiv:2305.18290 [cs].

Ren, M.; Cao, B.; Lin, H.; Liu, C.; Han, X.; Zeng, K.; Wan, G.; Cai, X.; and Sun, L. 2024. Learning or Self-aligning? Rethinking Instruction Fine-tuning. ArXiv:2402.18243 [cs].

Ruan, J.; Chen, Y.; Zhang, B.; Xu, Z.; Bao, T.; Du, G.; Shi, S.; Mao, H.; Li, Z.; Zeng, X.; and Zhao, R. 2023. TPTU: Large Language Model-based AI Agents for Task Planning and Tool Usage. arXiv:2308.03427.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761.

Snell, C.; Lee, J.; Xu, K.; and Kumar, A. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. ArXiv:2408.03314 [cs].

Tang, Q.; Deng, Z.; Lin, H.; Han, X.; Liang, Q.; Cao, B.; and Sun, L. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Wang, B.; Fang, H.; Eisner, J.; Van Durme, B.; and Su, Y. 2024a. LLMs in the Imaginarium: Tool Learning through Simulated Trial and Error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 10583–10604. Bangkok, Thailand: Association for Computational Linguistics.

Wang, Z.; Zeng, X.; Liu, W.; Li, L.; Wang, Y.; Shang, L.; Jiang, X.; Liu, Q.; and Wong, K.-F. 2024b. Toolflow: Boosting llm tool-calling through natural and coherent dialogue synthesis. *arXiv preprint arXiv:2410.18447*.

Weng, Y.; Zhu, M.; Xia, F.; Li, B.; He, S.; Liu, K.; and Zhao, J. 2023. Large Language Models are Better Reasoners with Self-Verification. ArXiv:2212.09561 [cs].

Xu, F.; Sun, Q.; Cheng, K.; Liu, J.; Qiao, Y.; and Wu, Z. 2024. Interactive Evolution: A Neural-Symbolic Self-Training Framework For Large Language Models. ArXiv:2406.11736 [cs].

Yan, F.; Mao, H.; Ji, C. C.-J.; Zhang, T.; Patil, S. G.; Stoica, I.; and Gonzalez, J. E. 2024. Berkeley Function Calling Leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Zeng, Y.; Ding, X.; Wang, Y.; Liu, W.; Ning, W.; Hou, Y.; Huang, X.; Qin, B.; and Liu, T. 2025. iTool: Reinforced Fine-Tuning with Dynamic Deficiency Calibration for Advanced Tool Use. *arXiv preprint arXiv:2501.09766*.

Zhang, J.; Lan, T.; Zhu, M.; Liu, Z.; Hoang, T.; Kokane, S.; Yao, W.; Tan, J.; Prabhakar, A.; Chen, H.; Liu, Z.; Feng, Y.; Awalgaonkar, T.; Murthy, R.; Hu, E.; Chen, Z.; Xu, R.; Niebles, J. C.; Heinecke, S.; Wang, H.; Savarese, S.; and Xiong, C. 2024. xLAM: A Family of Large Action Models to Empower AI Agent Systems. ArXiv:2409.03215 [cs].

Zhou, C.; Liu, P.; Xu, P.; Iyer, S.; Sun, J.; Mao, Y.; Ma, X.; Efrat, A.; Yu, P.; Yu, L.; Zhang, S.; Ghosh, G.; Lewis, M.; Zettlemoyer, L.; and Levy, O. 2023. LIMA: Less Is More for Alignment. ArXiv:2305.11206 [cs].