Try, Check and Retry: A Divide-and-Conquer Framework for Boosting Long-context Tool-Calling Performance of LLMs

Anonymous ACL submission

Abstract

Tool-calling empowers Large Language Models (LLMs) to interact with external environments. However, current methods often struggle to handle massive and noisy candidate tools in long-context tool-calling tasks, limiting their real-world application. To this end, we propose **Tool-DC**, a **D**ivide-and-**C**onquer framework for boosting tool-calling performance of LLMs. The core of Tool-DC is to reduce the reasoning difficulty and make full use of self-reflection ability of LLMs via a "Try-Check-Retry" paradigm. Specifically, Tool-DC involves two variants: 1) the training-free Tool-DC (TF), which is plug-and-play and flexible; 2) the training-based Tool-DC (TB), which is more inference-efficient. Extensive experiments show that both Tool-DC methods outperform their counterparts by a clear margin. Tool-DC (TF) brings up to **+25.10%** average gains against the baseline on BFCL and ACEBench benchmarks, while Tool-DC (TB) enables Qwen2.5-7B to achieve comparable or even better performance than proprietary LLMs, *e.g.*, OpenAI o3 and Claude-Haiku-4.5.

1 Introduction

While Large Language Models (LLMs) have shown remarkable successes (Ouyang et al., 2022; Achiam et al., 2023; Touvron et al., 2023), they often struggle to deal with the latest information and suffer from factual hallucinations, due to static parameters and closed knowledge boundaries. In response to this issue, the "*tool-calling*" paradigm has emerged, *i.e.*, the integration of external tools and application programming interfaces (APIs) enables LLMs to tackle complex, real-world scenarios (Schick et al., 2023; Qin et al., 2024).

The common tool-calling involves several processes. Specifically, given a large number of candidate tools, LLMs need to first select the appropriate tools and then formulate precise input arguments.

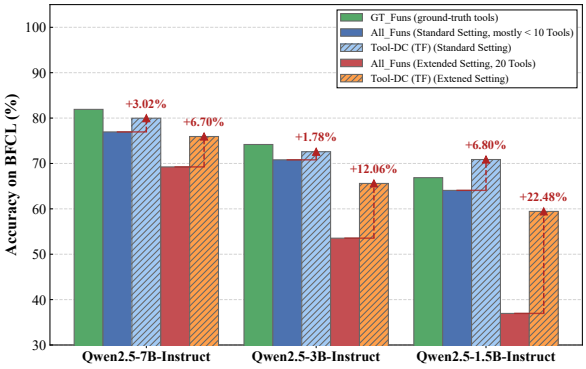

Figure 1: **Performance comparison on BFCL (Patil et al., 2025) with different candidate tool scales.** We see that as the number of candidate tools increases, all models' performance degrades significantly, whereas our Tool-DC method can effectively mitigate this issue.

Intuitively, the performance of tool-calling is sensitive to the number of candidate tools. As shown in our preliminary experiments (Figure 1), when scaling the number of candidate tools from <10 to 20, all LLMs suffer from performance degradation, especially for the smaller models. We conjecture that, on the one hand, ❶ *LLMs may fall short in dealing with long context information* caused by a large number of candidate tools. On the other hand, ❷ *confusing tools with similar semantics but different argument descriptions* could affect the argument-filling performance of LLMs.

There are some existing works involving analyzing and addressing these problems. For the first problem, a straightforward approach is to leverage an extra retriever to select the related tool subset (Qin et al., 2024; Patel et al., 2025). While effectively reducing the length of context, it highly relies on the performance of extra retriever (Robertson et al., 2009; Chen et al., 2024). More specifically, when the retriever fails in selecting the golden tools, LLMs struggle to predict the correct results due to the absence of golden tools in the context. For the second problem, Cui et al. (2025) reveal that many

tool-calling errors can be efficiently identified by the structured error checklists. Thus, they manually construct a global error checklist and use it to guide the precise tool-calling via an In-Context Learning (ICL) (Brown et al., 2020) manner. Although effective, the manually defined error checklist is not flexible and is difficult to cover all errors. Thus, there arises a question: *can we explore a more effective and efficient framework to boost the long-context tool-calling performance of LLMs?*

To achieve this goal, we propose **Tool-DC**, a **D**ivide-and-**C**onquer framework for better LLMs tool-calling, which contains two variants: 1) a training-free approach, Tool-DC (TF) and 2) a training-based approach, Tool-DC (TB). The core of Tool-DC is a *Try-Check-Retry* paradigm. Specifically, in the processes of Tool-DC (TF), we first split the total candidate tools into several groups and perform parallel tool-calling (***Try***), then verify the correctness of parallel inference results via strict schema constraints (***Check***), and lastly aggregate the validated results to refine global tool-calling accuracy (***Retry***). By doing so, we can not only reduce the length of context and reasoning difficulty, but also take full advantage of self-reflection capabilities of LLMs for better tool-calling. Moreover, considering the inference latency and deployment costs of Tool-DC (TF), we further propose to internalize the "Try-Check-Retry" decision capabilities into the model parameters. In practice, in Tool-DC (TB), we first construct a high-quality Chain-of-Thought (CoT) dataset by collecting the correct reasoning traces during the stage of Tool-DC (TF), and then use it to fine-tune the LLMs.

We conducted extensive experiments on two widely-used benchmarks, *i.e.*, BFCL (Patil et al., 2025) and ACEBench (Chen et al., 2025a), across multiple LLMs. Notably, considering that the number of candidate tools in the existing benchmarks is generally insufficient (<10), which is inconsistent with real-world scenarios, we introduce an *Extended Setting* by scaling the number of candidate tools to simulate real-world applications. Empirical results show that our Tool-DC (TF) consistently outperforms the vanilla tool-calling methods among all settings, especially for the small models in the extended setting, *i.e.*, up to **+25.10%** average gains. More encouragingly, with the help of Tool-DC (TB), Qwen2.5-7B (Yang et al., 2024) achieves an **83.16%** overall score on BFCL, surpassing the powerful OpenAI o3 and Claude-Haiku-4.5.

Contributions. To summarize, our contributions are three-fold: (1) We propose a divide-and-conquer framework (Tool-DC) for boosting the tool-calling performance of LLMs in the long-context and confusing tools scenarios. (2) Tool-DC provides two variants, where the training-free approach is plug-and-play and flexible, and the training-based approach has a higher inference efficiency. (3) Extensive experiments show that Tool-DC outperforms the vanilla method by a clear margin, *e.g.*, bringing up to **+25.10%** average gains against the training-free baseline.

2 Related Works

Equipping LLMs with external tools extends their capabilities to interact with real-world environments (Qu et al., 2025; He et al., 2025; Liao et al., 2025). However, the manner of providing all tool descriptions in the context is sub-optimal, as some candidate tools could be noisy and redundant (Chen et al., 2025b; Liang et al., 2025). There are many existing works aiming to improve the performance of tool-calling, which can be classified into training-free and training-based methods. The former optimize inference via retrieval and constraint injection but remain limited by the capabilities of the frozen base model (Chen et al., 2024; Zheng et al., 2024b; Hsieh et al., 2023; Ruan et al., 2023; Cui et al., 2025; Dang et al., 2025). The latter align models through supervised fine-tuning or reinforcement learning yet often suffer from high training costs and data synthesis bottlenecks (Huang et al., 2025; Hao et al., 2025; Zhang et al., 2025a; Qian et al., 2025; Zhang et al., 2025b). While effectively, these methods still struggle to deal with long-context tool-calling scenarios, where the number of candidate tools is large and many tools are similar but with different argument descriptions.

There are rare works involving addressing the above problems in the tool-calling field (Cui et al., 2025; Dang et al., 2025; Moon et al., 2024). Among which, Cui et al. (2025) attempt to mitigate tool-calling errors via manually designing a global error checklist. Despite its effectiveness, this method needs to carefully design a high-quality checklist, and still struggles with the long-context problem. Different from it, we propose a divide-and-conquer framework, which can effectively reduce the reasoning difficulty and make full use of the self-reflection capabilities of LLMs.

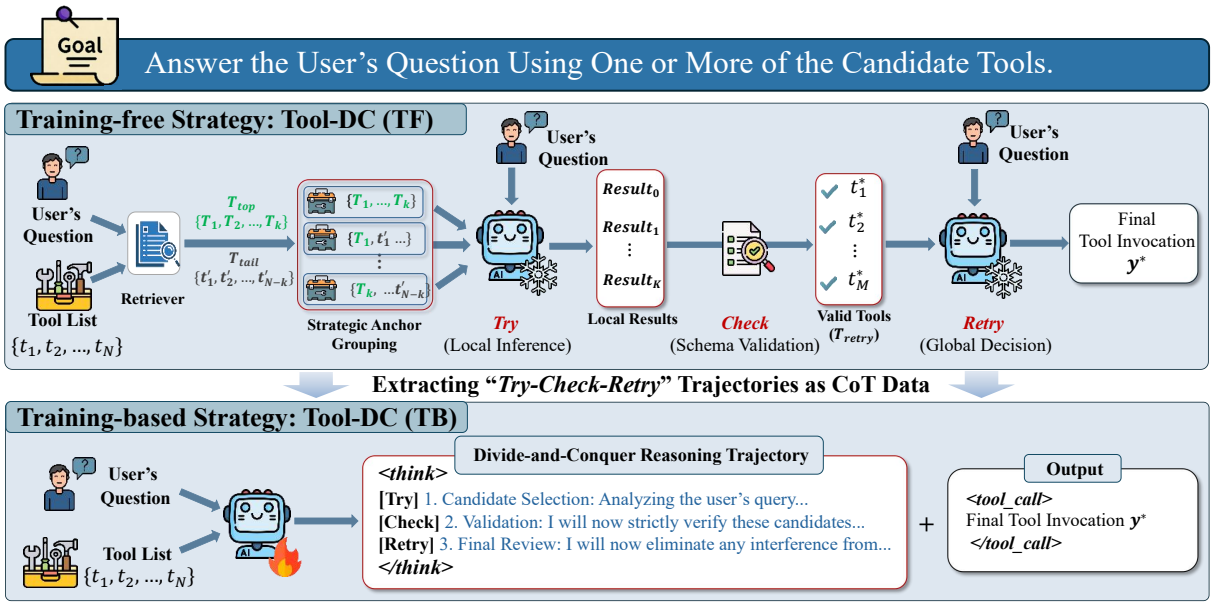

Figure 2: **Overview of our Tool-DC framework.** The training-free strategy employs a "Try-Check-Retry" pipeline to reduce the reasoning difficulty, while the training-based strategy leverages the prior reasoning trajectories to internalize this divide-and-conquer paradigm into model parameters via fine-tuning.

3 Methodology

We propose **Tool-DC**, a Divide-and-Conquer framework for boosting the long-context tool-calling performance of LLMs. As shown in Figure 2, Tool-DC contains two variants: 1) the training-free Tool-DC (TF), which reduces the reasoning difficulty via an explicit "Try-Check-Retry" pipeline; 2) the training-based Tool-DC (TB), which internalizes this divide-and-conquer paradigm into model parameters via fine-tuning.

3.1 Problem Formulation

Let q denote a user query and $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$ represent a library of N candidate tools. The objective of tool-calling is to enforce the model $\mathcal{M}$ to generate a sequence of tool invocations $\mathbf{y} = \{y_1, y_2, \ldots\}$, where each invocation $y_i = (t, \alpha)$ consists of a selected tool $t \in \mathcal{T}$ and its corresponding arguments α. A major challenge in real-world scenarios is the scale of N. Dealing with a vast search space makes directly modeling $P(\mathbf{y}|q, \mathcal{T})$ ineffective, as the substantial number of irrelevant candidates hinders the model's reasoning capabilities. In response to this problem, our Tool-DC aims to partition the global space $\mathcal{T}$ into manageable subspaces $\mathcal{S} = \{S_1, \ldots, S_K\}$. This decomposition strategy can reduce the decision complexity and facilitate precise tool-calling.

3.2 Tool-DC (TF): Training-free Approach

In this part, we introduce the details of our training-free approach, Tool-DC (TF), which involves the following three stages:

3.2.1 Try: Grouping and Local Inference

Considering that invoking tools from the global library $\mathcal{T}$ suffers from a low signal-to-noise ratio and interference among tools, in the first stage, we propose to isolate decision-making within low-noise subspaces via Strategic Anchor Grouping and Local Inference, which is illustrated in Figure 3.

Strategic Anchor Grouping. Similar to many prior works (Qin et al., 2024; Patel et al., 2025), we first retrieve Top-K relevant tools $\mathcal{T}_{top}$ via an extra retriever, *e.g.*, BM25 (Robertson et al., 2009). Then, we further construct K parallel groups $\{S_1, \ldots, S_K\}$ via the Strategic Anchor Grouping strategy. In practice, to obtain the subspace S_i, we take the i-th tool from $\mathcal{T}_{top}$ as an anchor and add a disjoint subset of distractors from $\mathcal{T}_{tail} = \mathcal{T} \setminus \mathcal{T}_{top}$. Such a grouping method can not only grant tools missed by the initial retrieval (contained in $\mathcal{T}_{tail}$) a chance to be evaluated, but also simultaneously decouple highly relevant candidates to prevent confusion. Notably, we preserve the original Top-K set as a distinct group $S_0 = \mathcal{T}_{top}$.

Local Inference. For each subspace $S_j \in \{S_0, S_1, \ldots, S_K\}$, the model $\mathcal{M}$ performs local inference and outputs an initial tool invocation or a

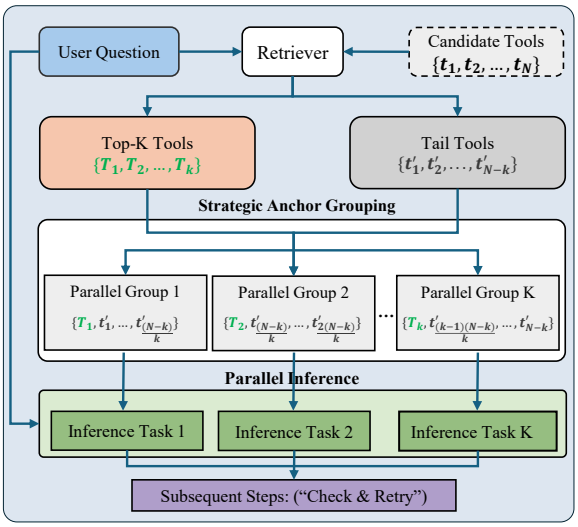

Figure 3: **Illustration of *Try* stage in Tool-DC (TF).** By splitting the total candidate tools into several parallel groups, Tool-DC (TF) can reduce the length of context and reasoning difficulty effectively.

null token:

$$o_j = \mathcal{M}(q, S_j), \tag{1}$$

where $j \in \{0, 1, \ldots, K\}$ denotes the index of subspace, and o_j represents the output (t_j, α_j) or the null token $\varnothing$. This process reduces the search space complexity and enables the model to generate more precise tool arguments.

3.2.2 Check: Schema Consistency Validation

Since the generated tool calls usually suffer from hallucinations, *e.g.*, invoking non-existent functions or hallucinating arguments, we leverage a rule-based *Consistency Validator* $\mathcal{F}(\cdot)$ to detect whether the tool calls are valid. Inspired by Cui et al. (2025), we design several schema constraints associated with the tools. In particular, the validity of each predicted output o_j is determined as:

$$\mathbb{I}_{\mathcal{C}}(o_j) = \begin{cases} 1 & \text{if } o_j \neq \varnothing \wedge \mathcal{F}(o_j, \text{Schema}(S_j)) \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where $\text{Schema}(\cdot)$ refers to the schema constraints that verify the validity from three dimensions:

- **Function Name Validity:** The name of invoked function t_j must explicitly exist in the defined tool set.
- **Argument Key Verification:** The keys in α_j must match the defined arguments, and all required parameters must be present.
- **Data Type Consistency:** The values assigned to arguments must adhere to the defined data types (e.g., `string`, `integer`, `boolean`).

As a result, this process yields a refined set of valid candidates $\mathcal{V} = \{o_j \mid \mathbb{I}_{\mathcal{C}}(o_j) = 1\}$.

3.2.3 Retry: Global Aggregation and Decision

Lastly, to take advantage of the self-reflection ability of LLMs, we introduce a refinement mechanism that utilizes the above validated candidates to obtain a more precise global decision. Specifically, we collect a refined candidate tool set $\mathcal{T}_{\text{retry}}$ by retrieving the original definitions of the tools presented in the valid set $\mathcal{V}$:

$$\mathcal{T}_{\text{retry}} = \{t \in \mathcal{T} \mid \exists (t, \alpha) \in \mathcal{V}\}. \tag{3}$$

The refined subset $\mathcal{T}_{\text{retry}}$ is provided in the context, which is then fed into the $\mathcal{M}$ to obtain the final tool invocation $\mathbf{y}^*$:

$$\mathbf{y}^* = \mathcal{M}(q, \mathcal{T}_{\text{retry}}). \tag{4}$$

In general, by splitting the total candidate tools into several subspaces and using the consistency validator to filter the invalid tools, we can obtain a more relevant subset of candidate tools. Furthermore, we employ the prior feedback to enforce the model to self-refine its output, thus leading to better tool-calling performance.

3.3 Tool-DC (TB): Training-based Approach

Although the proposed Tool-DC (TF) method is plug-and-play and flexible, it requires multiple forward passes and will lead to some inference latency. Thus, as an alternative, we additionally propose a training-based method, Tool-DC (TB), which aims to internalize the divide-and-conquer paradigm into model parameters via fine-tuning.

Data Construction. The key of Tool-DC (TB) is to construct the CoT training data. In practice, we mainly follow the pipeline of Tool-DC (TF) to collect the correct reasoning trajectories by using the xlam-function-calling-60k dataset (Zhang et al., 2025a) as raw training corpus $\mathcal{D}_{\text{raw}}$.To reduce dependence on external retrievers, we replace the group-based approach with an enumeration strategy, *i.e.*, the number of groups is equal to the number of all candidate tools ($K = N$). The pipeline of CoT data construction is shown in Algorithm 1. Notably, for ease of illustration, we simplify the reasoning template in the algorithm. The detailed template is shown in Figure 11 of Appendix A.7.

Algorithm 1 Data Construction for Tool-DC (TB)

Input: Raw training dataset $\mathcal{D}_{\text{raw}}$, library $\mathcal{T}$, model $\mathcal{M}$
Output: CoT training dataset $\mathcal{D}_{\text{CoT}}$
$\mathcal{D}_{\text{CoT}} \leftarrow \emptyset$
for $(q, \mathbf{y}_{\text{gt}}) \in \mathcal{D}_{\text{raw}}$ **do**
  $\mathcal{T}_{\text{valid}} \leftarrow \emptyset$
  // Phase 1: Try & Check
  for $t \in \mathcal{T}$ **do**
    $o \leftarrow \mathcal{M}(q, \{t\})$ ▷ Local inference
    // Check: Name, Args, and Type
    if $\mathbb{I}_{\mathcal{C}}(o) = 1$ **then**
      $\mathcal{T}_{\text{valid}} \leftarrow \mathcal{T}_{\text{valid}} \cup \{t\}$
    end if
  end for
  if $\mathcal{T}_{\text{valid}} = \emptyset$ **then continue**
  end if
  // Phase 2: Retry
  $\mathbf{y}^* \leftarrow \mathcal{M}(q, \mathcal{T}_{\text{valid}})$
  // Phase 3: Rationale Synthesis
  if $\mathbf{y}^* = \mathbf{y}_{\text{gt}}$ **then**
    Synthesize rationale R **via the template**:
      "1. Candidate Selection: ...
      2. Validation: ...
      3. Final Review: ..."
    CoT $\leftarrow$ `<think>`R`</think><tool>`y^*`</tool>`
    $\mathcal{D}_{\text{CoT}} \leftarrow \mathcal{D}_{\text{CoT}} \cup \{(q, \mathcal{T}, \text{CoT})\}$
  end if
end for
return $\mathcal{D}_{\text{CoT}}$

Optimization. After obtaining the CoT training dataset, denoted as $\mathcal{D}_{\text{CoT}}$, we can fine-tune the model $\mathcal{M}$ to minimize the negative log-likelihood of both reasoning traces and final tool invocation:

$$\mathcal{L} = -\sum_{(q,\mathcal{T})\in\mathcal{D}_{\text{CoT}}} \log P(\text{CoT} \mid q, \mathcal{T}). \quad (5)$$

By doing so, we can finally obtain the model that can self-reflect and generate the precise tool-calling in a single forward pass.

4 Experiments

4.1 Experimental Setup

Tasks and Datasets. We conduct the main experiments on two representative benchmarks: Berkeley Function-Calling Leaderboard (BFCL) (Patil et al., 2025) and ACEBench (Chen et al., 2025a). For BFCL, following standard evaluation protocols (Zhang et al., 2025b; Dang et al., 2025), we measure performance on *Non-Live* (synthetic) and *Live* (hand-crafted) subsets, respectively. For ACEBench, we adopt the *Normal (en)* split that covers diverse interaction scenarios as the test set.

Moreover, to assess robustness on different candidate tool scales, we employ two settings: 1) ***Standard Setting*** incorporates tool lists of the original benchmark to evaluate basic tool-calling capability; 2) ***Extended Setting*** expands candidates to 20 functions via randomly injecting irrelevant tools, simulating real-world noise. For evaluation, we report the strict Abstract Syntax Tree (AST) exact-match accuracy for both benchmarks. The details of all benchmarks are provided in Appendix A.1.

Models. We mainly evaluate the effectiveness of Tool-DC on Qwen2.5 (Yang et al., 2024) family (1.5B/3B/7B) models. To verify the universality of our methods, we also use the Qwen3-4B (Yang et al., 2025a), Llama-3.1/3.2 (Dubey et al., 2024) series (Llama-3.2-1B/-3B, Llama-3.1-8B), Gemma-3 (Team et al., 2025) series (1B/4B/12B) instruction models, and two closed-source LLMs, *i.e.*, GPT-4o-mini (Hurst et al., 2024) and DeepSeek-V3.2 (Liu et al., 2024). During the implementation of Tool-DC (TF), we set the number of groups to up to 5[1], *i.e.*, $K = min(5, N)$. For the extra retriever, we use the representative BM25 (Robertson et al., 2009) method. While for the implementation of Tool-DC (TB), we use the public `LLaMA-Factory` (Zheng et al., 2024a) toolkit as the training codebase. The details of training and inference hyperparameters are shown in Appendix A.2.

Compared Methods. We compare our Tool-DC with several cutting-edge counterparts. For training-free methods, we use the following baselines: 1) *GT_Funs*, only using the ground-truth tools as the context; 2) *All_Funs*, directly using all candidate tools as the context; 3) *Top-K*, using the retrieved Top-K relevant tools as the context; 4) *HiTEC-ICL* (Cui et al., 2025), using the manual designed error checklist to guide the precise tool-calling; 5) *ToolGT (Prompting)* (Dang et al., 2025), using a curriculum-inspired prompt to enforce step-by-step tool-calling. For training-based methods, in addition to the base model and vanilla supervised fine-tuning (SFT) baseline, *i.e.*, directly optimizing the model on ground-truth tool invocation without any reasoning traces, we also compare with several proprietary and tool-specialized LLMs. Due to space limitations, we provide the details of all compared models in Appendix A.3.

4.2 Main Results

4.2.1 Performance of Training-free Strategy

We first evaluate Qwen2.5 family models using our Tool-DC (TF) on BFCL and ACEBench benchmarks, and present the comparative results in Table 1. From these results, we can find that:

[1]The parameter analysis of K can be found in Section 4.4.

Table 1: **Performance comparison of Qwen2.5 models** with different training-free strategies on BFCL and ACEBench. The best and second-best results are in **bold** and underlined, respectively. The subscript results denote the relative performance gains against the All_Funs baseline.

Methods	Standard Setting					Extended Setting				
	BFCL			ACEBench	Avg.	BFCL			ACEBench	Avg.
	Non-Live	Live	Overall	Overall		Non-Live	Live	Overall	Overall	
Qwen2.5-1.5B-Instruct										
GT_Funs	70.96	62.77	66.87	47.92	57.40	70.96	62.77	66.87	47.92	57.40
All_Funs	69.96	58.18	64.07	46.33	55.20	40.69	33.23	36.96	22.00	29.48
Top-K	76.35	63.80	70.08	44.52	57.30$_{\uparrow 2.10}$	60.31	43.60	51.96	38.58	45.27$_{\uparrow 15.79}$
HiTEC-ICL	75.56	61.51	68.54	45.17	56.86$_{\uparrow 1.66}$	46.46	39.97	43.22	25.42	34.32$_{\uparrow 4.84}$
ToolGT (Prompting)	65.29	52.26	58.78	47.50	53.14$_{\downarrow 2.06}$	39.58	35.83	37.71	35.33	36.52$_{\uparrow 7.04}$
Tool-DC (TF) (Ours)	**77.63**	**63.43**	**70.53**	**49.08**	**59.81**$_{\uparrow 4.61}$	**64.13**	**62.00**	**63.07**	**46.08**	**54.58**$_{\uparrow 25.10}$
Qwen2.5-3B-Instruct										
GT_Funs	79.23	69.13	74.18	54.42	64.30	79.23	69.13	74.18	54.42	64.30
All_Funs	78.08	63.51	70.80	47.25	59.03	62.73	44.34	53.54	36.50	45.02
Top-K	78.00	66.03	72.02	45.78	58.90$_{\downarrow 0.13}$	63.73	46.71	55.22	38.02	46.62$_{\uparrow 1.60}$
HiTEC-ICL	**78.90**	63.80	71.35	49.33	60.34$_{\uparrow 1.31}$	63.79	45.23	54.51	34.92	44.72$_{\downarrow 0.30}$
ToolGT (Prompting)	68.94	57.22	63.08	**50.50**	56.79$_{\downarrow 2.24}$	59.29	54.92	57.11	46.58	51.85$_{\uparrow 6.83}$
Tool-DC (TF) (Ours)	78.54	**66.62**	**72.58**	49.42	**61.00**$_{\uparrow 1.97}$	**71.79**	**57.74**	**64.77**	**48.17**	**56.47**$_{\uparrow 11.45}$
Qwen2.5-7B-Instruct										
GT_Funs	85.81	78.02	81.92	65.17	73.55	85.81	78.02	81.92	65.17	73.55
All_Funs	**86.46**	67.44	76.95	59.92	68.44	75.71	62.77	69.24	58.58	63.91
Top-K	84.60	75.50	80.05	50.83	65.44$_{\downarrow 3.00}$	70.23	54.18	62.21	46.31	54.26$_{\downarrow 9.65}$
HiTEC-ICL	84.98	**76.24**	**80.61**	61.33	70.97$_{\uparrow 2.53}$	76.81	65.14	70.98	54.67	62.83$_{\downarrow 1.08}$
ToolGT (Prompting)	77.73	68.02	72.88	**68.92**	70.90$_{\uparrow 2.46}$	76.00	60.55	68.28	**62.42**	65.35$_{\uparrow 1.44}$
Tool-DC (TF) (Ours)	84.73	74.83	79.78	64.00	**71.89**$_{\uparrow 3.45}$	**84.40**	**70.00**	**77.20**	58.83	**68.02**$_{\uparrow 4.11}$

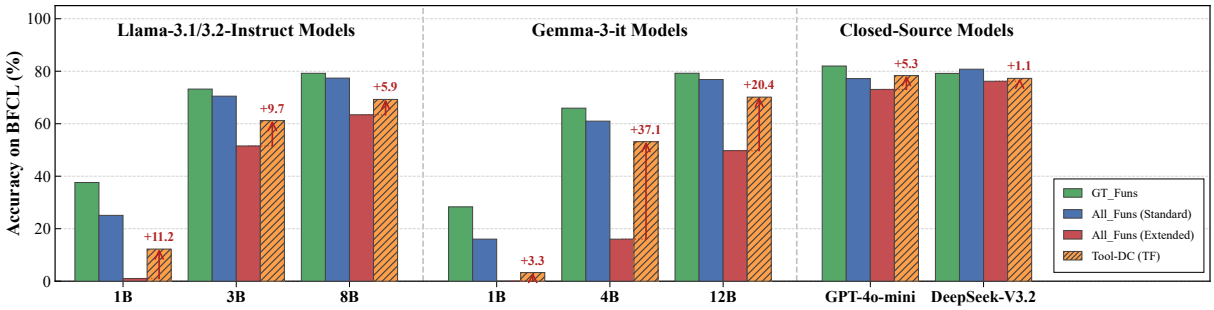

Figure 4: **Performance comparison of other LLMs** with different training-free strategies on the BFCL benchmark. Here, we mainly compare our Tool-DC (TF) method with the All_Funs baseline.

Tool-DC (TF) outperforms the baselines across model scales. First, in the Standard Setting, Tool-DC (TF) achieves the highest average scores across all Qwen2.5 family models, especially for the smaller models. Specifically, on the Qwen2.5-1.5B model, our method achieves an average score of 59.81%, outperforming the vanilla method by a significant margin, *i.e.*, +4.61% average gains. This indicates that smaller models struggle more to deal with long-context tool-calling tasks, while our methods can effectively reduce the reasoning difficulty and bring better performance.

Tool-DC (TF) exhibits strong robustness in noisy long-context tool-calling scenarios. Compared to the Standard Setting, we introduce more noisy candidate tools in the Extended Setting. As seen, in such a more difficult setting, all methods show performance degradation, especially in smaller models. Notably, for the All_Funs, the performance degradation is up to 25.72% average score. Conversely, by locating the relevant tools, our Tool-DC (TF) can alleviate the negative effect of noisy tools and maintain model performance effectively. Specifically, it outperforms the All_Funs by +25.10% average score in the Qwen2.5-1.5B model. These results prove the robustness of Tool-DC (TF).

Tool-DC (TF) brings consistent performance gains for various LLMs. In addition to Qwen2.5 models, we also evaluate Tool-DC (TF) on Llama3, Gemma3 and two closed-source LLMs, *i.e.*, GPT-4o-mini and DeepSeek-V3.2. Figure 4 illustrates the comparative results on the BFCL benchmark,

Table 2: **Performance comparison between our tuned models and other proprietary models on BFCL**. Notably, the results of proprietary models are from the official leaderboard of BFCL.

Models	Non-Live	Live	Overall
Close-source Models			
OpenAI o3	81.94	73.21	77.58
DeepSeek-V3.2-Exp	85.52	76.02	80.77
Claude-Haiku-4.5	86.50	78.68	82.59
Gemini-3-Pro-Preview	**90.65**	**83.12**	**86.89**
Open-source Models			
Large-scale Models			
Qwen3-235B-A22B-Inst	90.12	76.61	83.37
Llama-3.3-70B-Inst	88.02	76.61	82.32
Tool-specialized Models			
xLAM-2-8b-fc-r	84.58	67.95	76.27
ToolACE-DEV-8B	89.67	75.20	82.44
ToolACE-MT-8B	84.94	71.52	78.23
Hammer2.1-3B	85.50	69.50	77.50
Hammer2.1-7B	84.96	70.54	77.75
Tuned Models			
Qwen2.5-7B-Instruct	86.46	67.44	76.95
-w Vanilla SFT	84.23	74.24	79.24$_{\uparrow 2.29}$
-w Tool-DC (TB)	86.67	79.64	83.16$_{\uparrow 6.21}$
Llama-3.2-3B-Instruct	82.67	58.33	70.50
-w Vanilla SFT	83.31	62.99	73.15$_{\uparrow 2.65}$
-w Tool-DC (TB)	83.17	63.95	73.56$_{\uparrow 3.06}$

from which we find that our Tool-DC (TF) consistently boosts the tool-calling performance across all base models. Notably, for the powerful GPT-4o-mini model, Tool-DC (TF) can still lead to +5.3% performance gains, proving its effectiveness.

4.2.2 Performance of Training-based Strategy

The comparative results of different tuned models on ACEBench and BFCL benchmarks are presented in Figure 5 and Table 2, respectively. Notably, due to limited computing resources, we only adopt Tool-DC (TB) to fine-tune some representative models. From them, we can observe that:

Tool-DC (TB) yields consistent gains across diverse models. In addition to the results on Qwen2.5 family models, we also apply our Tool-DC (TB) to boost the tool-calling performance of Llama3 and Qwen3 models. Taking the ACEBench benchmark as an example, we find that Tool-DC (TB) brings +24.95% performance gain for Llama-3.2-3B-Instruct model, and +15.9% for Qwen3-4B-Instruct model. These results can prove the universality of our Tool-DC (TB) method.

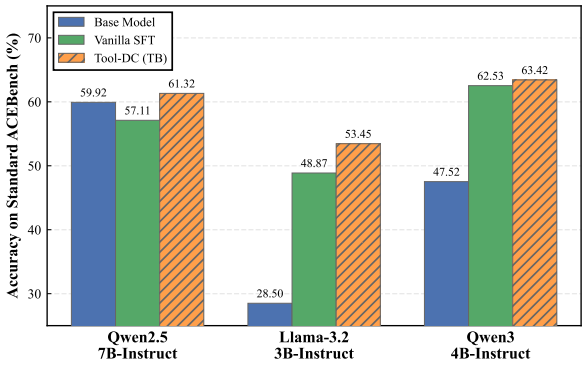

Figure 5: Comparison between base models and tuned models using Tool-DC (TB) on Standard ACEBench.

Table 3: **Ablation study on Tool-DC (TF)**. Here, we use the Qwen2.5-3B-Instruct as the base model and evaluate on the Extended Setting of BFCL.

Methods	Modules			Performance		
	Try	Check	Retry	Non-Live	Live	Overall
Base	✗	✗	✗	62.73	44.34	53.54
Tool-DC (TF)	✓	✓	✓	**71.79**	**57.74**	**64.77**
w/o Try	✗	✓	✓	44.19	29.39	36.79
w/o Check	✓	✗	✓	52.06	52.41	52.24
w/o Retry	✓	✓	✗	7.33	3.18	5.26

Our tuned models achieve remarkable performance against the proprietary LLMs. In Table 2, we report the official results from the leaderboard[2] of BFCL for the proprietary LLMs. It can be seen, our Tool-DC (TB) can not only improve the tool-calling performance for various base models, but also enable them to achieve remarkable (or even better) performance against the other proprietary LLMs. More specifically, with the help of Tool-DC (TB), Qwen2.5-7B-Instruct achieves 83.16% overall score on BFCL, outperforming the OpenAI o3, DeepSeek-V3.2, and Claude-Haiku-4.5. These results demonstrate the superiority of our method.

4.3 Ablation Study

In this part, we conduct ablation studies to investigate the effect of different stages (*Try*, *Check*, and *Retry*) of Tool-DC (TF) method, respectively. Notably, we use the Qwen2.5-3B-Instruct as the base model, and report the results on the Extended Setting of BFCL in Table 3. As seen, removing any stage in Tool-DC (TF) will lead to performance degradation, proving their effectiveness.

Firstly, when removing the *Try* stage, the average performance of Qwen2.5-3B-Instruct drops from 64.77% to 36.79%, indicating the importance

[2]https://gorilla.cs.berkeley.edu/leaderboard.html

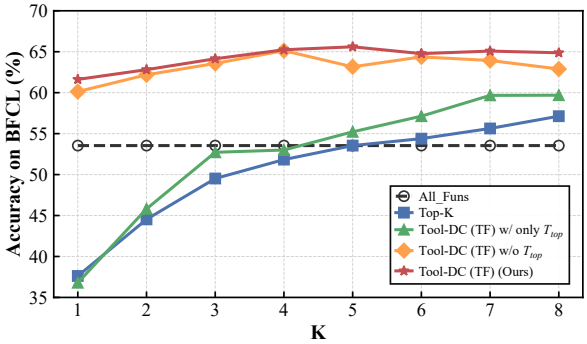

Figure 6: **Impact of Group Count** K. We evaluate Qwen2.5-3B-Instruct using different training-free methods on the Extended Setting of BFCL.

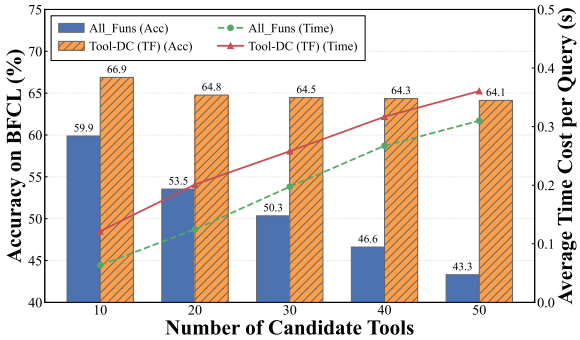

Figure 7: **Scalability and Efficiency Analysis.** Here, we use the Qwen2.5-3B-Instruct as the base model, and compare our Tool-DC (TF) with the All_Funs baseline method on the Extended Setting of BFCL.

of task decomposition. Without the divide-and-conquer mechanism, Tool-DC (TF) falls short in checking its outputs and self-reflection. Secondly, when removing the *Check* stage, the refinement process tends to be unstable and lead to sub-optimal results. Lastly, when removing the *Retry* stage, we observe a performance collapse, *i.e.*, from 64.77% to 5.26%. This highlights: after filtering out potentially correct answers via *Check*, Tool-DC (TF) relies on *Retry* to leverage these positive signals. Without *Retry*, Tool-DC (TF) cannot process the high-potential candidates identified by *Check*, resulting in the loss of valid solutions.

4.4 More Analyses

Sensitivity analysis of Group Count K**.** We investigate the impact of the group count K, which is an important hyperparameter in the first stage of Tool-DC (TF). In practice, we conduct comparative experiments on the Extended Setting of BFCL by ranging K from 1 to 8. As shown in Figure 6, the Top-K baseline method suffers from significant performance drops at smaller K (<3) due to aggressive sample filtering. Conversely, by making full use of all useful candidate tools, our Tool-DC (TF) exhibits superior robustness to the number of K. More specifically, in Figure 6, "Tool-DC (TF) w/ only $\mathcal{T}_{top}$" denotes that we only use the retrieved Top-K tools in the *Try* stage, which can be seen as an improvement on the Top-K method via our *Check* and *Retry* stages. By doing so, Tool-DC (TF) consistently improves the performance of the Top-K baseline, indicating the effectiveness of the self-reflection mechanism. Moreover, when K is set to 5, our Tool-DC (TF) method performs best, thus leaving it as the default setting in our work.

Scalability and efficiency analysis. To better simulate the real-world scenarios where the candidate tools are massive and noisy, we increase the number of candidate tools N from 10 to 50 for the BFCL benchmark by randomly adding some irrelevant tools. We compare the results of the Qwen2.5-3B-Instruct model by using the All_Funs and our Tool-DC (TF), respectively. As illustrated in Figure 7, the performance of All_Funs baseline drops significantly as the N increases, *i.e.*, from 59.88% to 43.30%. On the contrary, our Tool-DC (TF) can maintain the performance effectively, demonstrating its robustness. Furthermore, we additionally compare the inference latency between Tool-DC (TF) and All_Funs. Although Tool-DC (TF) indeed leads to some inference overhead, it is tolerable against the performance gains of Tool-DC (TF).

5 Conclusion

In this paper, we propose Tool-DC, a divide-and-conquer framework to boost the tool-calling performance of LLMs via mitigating the negative effect of long-context and noisy candidate tools. Specifically, Tool-DC involves two variants: the training-free Tool-DC (TF) and training-based Tool-DC (TB). The former approach leverages the "Try-Check-Retry" pipeline to reduce the reasoning difficulty and make full use of the self-reflection ability of LLMs, thus leading to precise tool-calling. The latter approach aims to internalize the divide-and-conquer paradigm into model parameters via fin-tuning. Extensive experiments show that our Tool-DC consistently outperforms both training-free and training-based counterparts across various model scales and architectures.

Limitations

Our work has several potential limitations that present opportunities for future research. First, we only used xlam-function-calling-60k as the seed dataset to construct our CoT data for Tool-DC (TB). Although this dataset is large, it lacks sufficient diversity and contains a limited number of tools in the context. In the future, we plan to construct a more diverse training set with massive and noisy context to simulate real-world scenarios, and explore reinforcement learning, *e.g.*, GRPO (Guo et al., 2025), to further improve the model performance. Secondly, our current experiments are limited to single-step tool-calling tasks, and we do not evaluate our framework on multi-step nested scenarios, such as those introduced in the latest version of BFCL. Therefore, future work could extend our method to more complex tool-calling settings.

Ethics Statement

We take ethical considerations very seriously and strictly adhere to the ACL Ethics Policy. This paper proposes a method to boost the tool-calling performance of LLMs in long-context scenarios. This method involves both training-free and training-based strategies, using a divide-and-conquer mechanism to assist model decision-making, rather than encouraging them to learn privacy-sensitive knowledge that may raise ethical concerns. Moreover, all training and evaluation datasets used in this paper are publicly available and have been widely adopted by researchers. Thus, we believe that this research will not pose ethical issues.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2025. Introducing claude haiku 4.5. https://www.anthropic.com/news/claude-haiku-4-5.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in neural information processing systems*, pages 1877–1901.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, and Wu Liu. 2025a. ACEBench: A comprehensive evaluation of LLM tool usage. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12970–12998.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2318–2335.

Yi-Chang Chen, Po-Chun Hsu, Chan-Jan Hsu, and Dashan Shiu. 2025b. Enhancing function-calling capabilities in llms: Strategies for prompt formats, data integration, and multilingual translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pages 99–111.

Yue Cui, Liuyi Yao, Shuchang Tao, Weijie Shi, Yaliang Li, Bolin Ding, and Xiaofang Zhou. 2025. Enhancing tool learning in large language models with hierarchical error checklists. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16357–16375.

Hy Dang, Tianyi Liu, Zhuofeng Wu, Jingfeng Yang, Haoming Jiang, Tao Yang, Pei Chen, Zhengyang Wang, Helen Wang, Huasheng Li, et al. 2025. Improving large language models function calling and interpretability via guided-structured templates. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 24437–24453.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Bingguang Hao, ZengZhuang Xu, Maolin Wang, Yuntao Wen, Yicheng Chen, Cunyin Peng, Long

Chen, Dong Wang, Xiangyu Zhao, Jinjie Gu, et al. 2025. Funreason: Enhancing large language models' function calling via self-refinement multiscale loss and automated data refinement. *arXiv preprint arXiv:2505.20192*.

Jie He, Jennifer Neville, Mengting Wan, Longqi Yang, Hui Liu, Xiaofeng Xu, Xia Song, Jeff Z. Pan, and Pei Zhou. 2025. GenTool: Enhancing tool generalization in language models through zero-to-one and weak-to-strong simulation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1097–1122.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Xu Huang, Weiwen Liu, Xingshan Zeng, Yuefeng Huang, Xinlong Hao, Yuxian Wang, Yirong Zeng, Chuhan Wu, Yasheng Wang, Ruiming Tang, et al. 2025. Toolace-dev: Self-improving tool learning via decomposition and evolution. *arXiv preprint arXiv:2505.07512*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Yibin Lei, Liang Ding, Yu Cao, Changtong Zan, Andrew Yates, and Dacheng Tao. 2023. Unsupervised dense retrieval with relevance-aware contrastive pre-training. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10932–10940.

Xiaobo Liang, Wenjin Xie, Juntao Li, Wanfu Wang, Yibin Chen, Kehai Chen, and Min Zhang. 2025. Tool learning via inference-time scaling and cycle verifier. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24658–24671.

Yusheng Liao, Shuyang Jiang, Yanfeng Wang, and Yu Wang. 2025. Reflectool: Towards reflection-aware tool-augmented clinical agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13507–13531.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. 2025. Robust function-calling for on-device language model via function masking. In *The Thirteenth International Conference on Learning Representations*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. 2024. Efficient and scalable estimation of tool representations in vector space. *arXiv preprint arXiv:2409.02141*.

OpenAI. 2025. Openai-o3. https://openai.com/index/introducing-o3-and-o4-min.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 27730–27744.

Bhrij Patel, Davide Belli, Amir Jalalirad, Maximilian Arnold, Aleksandr Ermovol, and Bence Major. 2025. Dynamic tool dependency retrieval for efficient function calling. *arXiv preprint arXiv:2512.17052*.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. 2025. A new era of intelligence with gemini 3. https://deepmind.google/models/gemini/pro/.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

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

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, et al. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 68539–68551.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. 2025b. Qwen2. 5-1m technical report. *arXiv preprint arXiv:2501.15383*.

Xingshan Zeng, Weiwen Liu, Xu Huang, Zezhong Wang, Lingzhi Wang, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruiming Tang, et al. 2025. Toolace-r: Tool learning with adaptive self-refinement. *arXiv preprint arXiv:2504.01400*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, and Haolin Chen. 2025a. xLAM: A family of large action models to empower AI agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025b. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, YeYanhan YeYanhan, and Zheyan Luo. 2024a. 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)*, pages 400–410.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024b. Toolrerank: Adaptive and hierarchy-aware reranking for tool retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16263–16273.

A Appendix

A.1 Details of Benchmarks

The Berkeley Function-Calling Leaderboard (BFCL). Patil et al. (2025) introduce a comprehensive benchmark BFCL for evaluating function-calling capabilities of LLMs, comprising two primary evaluation categories: Non-Live and Live. Each category further includes four distinct task types: single, multiple, parallel, and parallel multiple function invocations. The total number of samples in the used test set is 2,501.

- **Single**: LLMs are provided with a single function description and must generate exactly one corresponding call.
- **Multiple**: LLMs are provided with a set of candidate functions and must select and invoke the most appropriate one.
- **Parallel**: LLMs need to generate multiple function calls simultaneously for a single query, determining the appropriate number of invocations.
- **Parallel Multiple**: Combining the challenges of *Parallel* and *Multiple*, LLMs need to select from several candidates to invoke zero or more functions as required.

ACEBench. ACEBench (Chen et al., 2025a) is a diverse tool-calling benchmark that offers fine-grained coverage across a broad range of domains. Specifically, it encompasses 8 major domains and 68 sub-domains, including but not limited to technology, finance, entertainment, society, health, culture, and environment. The benchmark is organized

into three primary categories: Normal, Special, and Agent. In this work, we focus exclusively on the **Normal** category, which contains 828 test samples. The Normal category is further subdivided into the following five tasks: *Single-Turn*, *Multi-Turn*, *Similar APIs*, *Profile*, and *Atom*.

A.2 Training and Inference Details

In this work, we use the xlam-function-calling-60k dataset (Zhang et al., 2025a) as raw training corpus, and construct a CoT training set with 46,897 samples. Table 4 shows the detailed training hyperparameters. Specifically, we set the batch size to 16 and the peak learning rate to 2e-5. All models are trained with LoRA (Hu et al., 2021) for 2 epochs on two NVIDIA A800 80GB GPUs, requiring approximately 10 hours for finetuning 7B models using our Tool-DC (TB). For inference, we use the VLLM[3] engine for accelerating the model generation, and set the temperature to 0, the maximum length of output tokens to 512.

Table 4: Training hyperparameters for all models.

Hyperparameter	Value
Batch Size	16
Learning Rate	2e-5
LR Scheduler	cosine
Warmup Ratio	0.1
Epochs	2

A.3 Details of Compared Models

In our training-based experiments, we compare our tuned models with both general-purpose commercial APIs (OpenAI o3[4] (OpenAI, 2025), DeepSeek-V3.2-Exp[5] (Liu et al., 2025), Claude-Haiku-4.5[6] (Anthropic, 2025), and Gemini-3-Pro-Preview[7] (Pichai et al., 2025)) and open-source large-scale models (Qwen3-235B-A22B-Inst (Yang et al., 2025a) and Llama-3.3-70B-Inst (Dubey et al., 2024)), as well as some specialized tool-calling models, including ToolACE-DEV-8B (Huang et al., 2025), ToolACE-MT-8B (Zeng et al., 2025), xLAM-2 (Zhang et al., 2025a), and Hammer2.1(1.5B/3B/7B) (Lin et al., 2025).

[3]https://github.com/vllm-project/vllm
[4]https://openai.com/index/introducing-o3-and-o4-mini
[5]https://api-docs.deepseek.com/news/news250929
[6]https://www.anthropic.com/news/claude-haiku-4-5
[7]https://deepmind.google/models/gemini/pro/

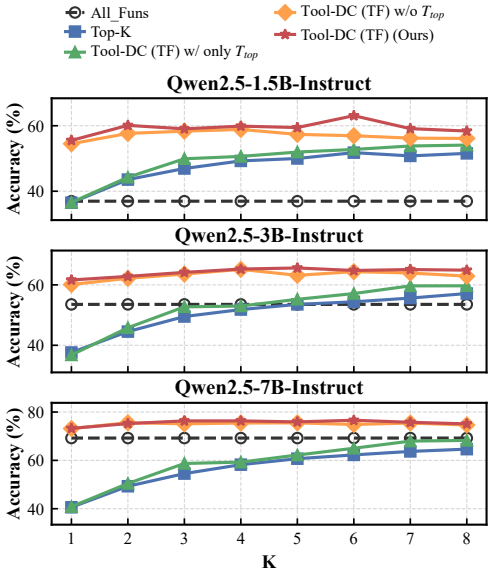

Figure 8: Sensitivity analysis of group count K across Qwen2.5 series. All models are evaluated on the Extended Setting of BFCL.

A.4 Sensitivity Analysis of Group Count K

We extend the sensitivity analysis of group count K to Qwen2.5-1.5B, 3B, and 7B-Instruct. Results in Figure 8 consistently show that while baselines like *Top-K* suffer significant drops at low K (< 3) due to over-filtering, our method remains robust even at $K = 1$. Performance stabilizes at $K \in [4, 6]$, confirming that a moderate partition size effectively balances noise reduction and context completeness across varying model scales.

A.5 Performance on Long-context LLMs

To verify whether the long-context tool-calling issue can be fixed by scaling the context windows, we evaluate the cutting-edge long-context LLMs, including Qwen2.5-7B-Instruct-1M (Yang et al., 2025b) and InternLM2.5-7B-Chat-1M (Cai et al., 2024), which have up to 1M tokens in the context windows, on the BFCL benchmark. Specifically, we mainly conduct experiments on the Extended Settings, and report the comparative results in Table 5. For reference, we also report the results of GT_Funs as the upper-bound.

As seen, while InternLM2.5-7B-Chat-1M has a long context window, it still suffers from dramatic performance drops in the Extended Setting, indicating that simply scaling context windows cannot address the long-context tool-calling problem. This underscores the necessity of exploring a more robust tool-calling method. Encouragingly, with the help of our Tool-DC (TF), InternLM2.5-7B-Chat-

Table 5: Performance of long-context LLMs on the Extended Setting of BFCL.

Methods	Non-Live				Live				Overall		
	Simple	Multiple	Parallel	Parallel Multiple	Simple	Multiple	Parallel	Parallel Multiple	Non-Live	Live	Overall
Qwen2.5-7B-Instruct-1M											
GT_Funs	74.92	95.00	90.50	85.50	80.23	77.30	62.50	66.67	86.48	77.50	81.99
All_Funs	68.67	85.50	84.00	71.00	63.18	65.81	62.50	58.33	77.29	65.14	71.22
Tool-DC (TF) (Ours)	73.17	89.00	88.50	79.50	65.89	69.71	62.50	66.67	82.54	68.84	75.69
InternLM2.5-7B-Chat-1M											
GT_Funs	51.92	85.00	70.50	51.00	55.81	51.28	25.00	20.83	64.60	51.30	57.95
All_Funs	17.92	26.00	25.00	10.50	14.34	12.44	18.75	4.17	19.85	12.73	16.29
Tool-DC (TF) (Ours)	41.00	65.00	46.00	28.50	35.66	32.19	12.50	12.50	45.12	32.27	38.70

1M can alleviate the performance drop effectively, *i.e.*, from 16.29% to 38.70%, continuing to prove its effectiveness and superiority.

A.6 Performance on More Retrievers

In our work, we use the representative unsupervised retrieval method BM25 as the base retriever. Here, we further conduct experiments to investigate the generality of Tool-DC on the other retrievers. Specifically, two cutting-edge dense retrievers, including bge-reranker-v2-m3 (Chen et al., 2024) and ToolBench_IR (Qin et al., 2024), are used in this experiment. We evaluate the Qwen2.5-3B-Instruct model on the Extended Setting of BFCL and report the comparative results in Table 6. From these results, we can find that our Tool-DC (TF) brings consistent performance gains against the Top-K baseline across all base retrievers, proving its generality. Notably, although bge-reranker-v2-m3 is effective, it is a two-tower BERT (Devlin et al., 2019) model, which is inference-inefficient. While ToolBench_IR is finetuned on ToolBench, it is difficult to generalize to other tool-calling tasks, thus leading to sub-optimal results. Moreover, considering BM25 is a powerful and efficient unsupervised retriever (Lei et al., 2023), we finally choose it as the base retriever in our experiment.

A.7 Prompt Details

Here, we provide the detailed prompts used in our Tool-DC framework. Specifically, for Tool-DC (TF), the "Try" and "Retry" prompts are presented in Figure 9 and Figure 10, respectively. For Tool-DC (TB), we present the prompt template of data construction in Figure 11.

A.8 Full Results

Due to space limitations, we only report the averaged experimental results in the main body of this paper. Here, we present the detailed results to help readers better reproduce our main experiments. Specifically, the detailed results of training-free approaches on BFCL and ACEBench benchmarks are provided in Table 7 and Table 9, respectively. The detailed results of training-based approaches on BFCL and ACEBench benchmarks are provided in Table 8 and Table 10, respectively.

A.9 AI Assistant Usage

During the paper writing, we used a proprietary LLM, *i.e.*, Gemini-3-Pro, as the general-purpose assistant to polish some sentences. We state that we did not prompt them to generate research ideas or search for related works, thus avoiding the risk of introducing false information.

Table 6: Performance of Qwen2.5-3B-Instruct using different retrievers on the Extended Setting of BFCL.

Methods	Non-Live				Live				Overall		
	Simple	Multiple	Parallel	Parallel Multiple	Simple	Multiple	Parallel	Parallel Multiple	Non-Live	Live	Overall
GT_Funs	70.92	91.50	79.00	75.50	67.83	70.37	56.25	37.50	79.23	69.13	74.18
All_Funs	64.42	71.00	62.50	53.00	43.41	44.92	43.75	29.17	62.73	44.34	53.54
bge-reranker-v2-m3											
-w Top-K	67.42	83.50	74.50	61.00	52.71	64.10	50.00	37.50	71.60	61.29	66.45
-w Tool-DC (TF)	68.08	82.50	75.50	68.50	57.75	64.29	50.00	41.67	73.65	62.47	68.06
ToolBench_IR											
-w Top-K	63.42	73.00	66.00	41.50	42.25	47.10	50.00	37.50	60.98	46.04	53.51
-w Tool-DC (TF)	68.83	81.50	74.00	58.50	53.10	62.01	56.25	37.50	70.71	59.81	65.26
BM25											
-w Top-K	64.42	74.50	71.00	45.00	41.09	48.53	43.75	29.17	63.73	46.71	55.22
-w Tool-DC (TF)	64.67	81.00	77.50	64.00	48.84	60.40	43.75	45.83	71.79	57.74	64.77

Prompt template of "Try" stage in Tool-DC (TF)

```
# System:
You are a Function Selection Expert. Your task is to identify ALL functions that are semantically
relevant to the user's question from the provided list. Extract information from the user's
question and substitute it into the function parameters.
Read the user's question and the function descriptions carefully. Choose any function that could
potentially meet user needs or meet a part of user needs.
If you decide to invoke any of the function(s), you MUST put it in the format of:
[func_name1(params_name1=params_value1...), func_name2(params)]. You SHOULD NOT include any other
text in the response.
Here is a list of functions in json format that you can invoke.
<Tools of Each Group>
# User:
<Question>
```

Figure 9: The prompt template of "Try" stage in Tool-DC (TF).

Prompt template of "Retry" stage in Tool-DC (TF)

```
# System:
You are an expert in composing functions. You are given a question and a set of possible functions.
Based on the question, you will need to make one or more function/tool calls to achieve the
purpose. If none of the functions can be used, point it out. If the given question lacks the
parameters required by the function, also point it out.
You should only return the function calls in your response. If you decide to invoke any of
the function(s), you MUST put it in the format of: [func_name1(params_name1=params_value1...),
func_name2(params)]. You SHOULD NOT include any other text in the response.
At each turn, you should try your best to complete the tasks requested by the user within the
current turn. Continue to output functions to call until you have fulfilled the user's request
to the best of your ability.
Here is a list of functions in json format that you can invoke. <Tools>
# User:
<Question>
```

Figure 10: The prompt template of "Retry" stage in Tool-DC (TF).

Prompt template for constructing the CoT data

```
# System:
You are an expert in composing functions. You are given a question and a set of possible functions.
Based on the question, you will need to make one or more function/tool calls to achieve the
purpose. If none of the function can be used, point it out. If the given question lacks the
parameters required by the function, also point it out.
You should only return the function call in tools call sections. Continue to output functions to
call until you have fulfilled the user's request to the best of your ability.
Here is a list of functions in json format that you can invoke. <Tools>
For each interaction, you MUST strictly follow this two-step process:
Step 1: Reasoning (<think>)
You must engage in a detailed chain-of-thought enclosed within <think></think> tags.
This process must follow these exact 3 sub-steps:
Candidate Selection: Analyze the user's question. Iterate through the function list...
Validation: Strictly check the candidate list against function definitions...
Final Review: Focus exclusively on the effective candidate list...
Step 2: Invoke (<tool_call>)
If you decide to invoke function(s), output them in the following specific format:
<tool_call>[func_name1(params_name1=value1, ...), func_name2(params)]</tool_call>
# User:
<Question>
# Assistent:
<think>
1. Candidate Selection: Analyzing the user's query, I will attempt to map key information to the
function parameters. The functions that potentially match and may have their parameters filled
are: <candidate_tool_calls>
2. Validation: I will now strictly verify these candidates against their definitions, ensuring
all parameter types and constraints are met. The functions that pass this strict verification
are: <valid_tools>.
3. Final Review: I will now eliminate any interference from irrelevant functions and focus solely
on the valid candidates.
</think>
<tool_call><final_tool_invocation></tool_call>
```

Figure 11: The prompt template for constructing the CoT data in Tool-DC (TB).

Table 7: Detailed results of different training-free methods on BFCL, which is a full version of Table 1.

Methods	Non-Live				Live				Overall		
	Simple	Multiple	Parallel	Parallel Multiple	Simple	Multiple	Parallel	Parallel Multiple	Non-Live	Live	Overall
Standard Setting											
Qwen2.5-1.5B-Instruct											
GT_Funs	63.33	88.50	68.50	63.50	65.12	63.25	43.75	29.17	70.96	62.77	66.87
All_Funs	63.33	84.00	68.00	64.50	65.12	57.26	43.75	33.33	69.96	58.18	64.07
Top-K	66.42	83.50	79.50	76.00	68.99	63.15	50.00	45.83	76.35	**63.80**	70.08
HiTEC-ICL	**67.75**	83.50	79.00	72.00	**70.54**	59.83	**56.25**	41.67	75.56	61.51	68.54
ToolGT (Prompting)	66.67	78.00	64.00	52.50	67.05	49.29	25.00	41.67	65.29	52.26	58.78
Tool-DC (TF) (Ours)	66.50	**85.50**	**80.50**	**78.00**	68.60	62.68	50.00	**50.00**	**77.63**	63.43	**70.53**
Qwen2.5-3B-Instruct											
GT_Funs	70.92	91.50	79.00	75.50	67.83	70.37	56.25	37.50	79.23	69.13	74.18
All_Funs	70.83	88.00	79.50	74.00	68.22	63.06	56.25	37.50	78.08	63.51	70.80
Top-K	71.00	89.50	78.00	73.50	67.05	66.29	**62.50**	45.83	78.00	66.03	72.02
HiTEC-ICL	**72.58**	87.50	**80.50**	**75.00**	**68.60**	63.25	**62.50**	37.50	**78.90**	63.80	71.35
ToolGT (Prompting)	67.75	81.50	68.50	58.00	**68.60**	55.46	25.00	33.33	68.94	57.22	63.08
Tool-DC (TF) (Ours)	70.67	**90.00**	78.50	**75.00**	67.05	**66.95**	**62.50**	**50.00**	78.54	**66.62**	**72.58**
Qwen2.5-7B-Instruct											
GT_Funs	72.25	96.00	91.00	84.00	77.13	78.73	62.50	66.67	85.81	78.02	81.92
All_Funs	75.33	94.50	91.50	84.50	76.74	74.93	62.50	70.83	86.46	67.44	76.95
Top-K	71.92	93.50	**91.00**	82.00	77.13	75.40	**62.50**	70.83	84.60	75.50	80.05
HiTEC-ICL	**72.42**	**95.00**	89.00	**83.50**	**78.29**	**76.07**	**62.50**	70.83	**84.98**	**76.24**	**80.61**
ToolGT (Prompting)	69.92	82.50	84.00	74.50	**78.29**	65.53	56.25	**75.00**	77.73	68.02	72.88
Tool-DC (TF) (Ours)	71.92	94.00	**91.00**	82.00	77.91	74.93	**62.50**	66.67	84.73	75.20	79.97
Extended Setting											
Qwen2.5-1.5B-Instruct											
GT_Funs	63.33	88.50	68.50	63.50	65.12	63.25	43.75	29.17	70.96	62.77	66.87
All_Funs	51.75	49.00	35.50	26.50	32.95	33.90	25.00	12.50	40.69	33.23	36.96
Top-K	61.75	**69.50**	69.50	**40.50**	40.70	45.20	25.00	16.67	60.31	43.60	51.96
HiTEC-ICL	57.33	53.00	45.50	30.00	41.09	40.36	31.25	16.67	46.46	39.97	43.22
ToolGT (Prompting)	46.83	55.00	31.00	25.50	38.76	36.37	0.00	4.17	39.58	35.83	37.71
Tool-DC (TF) (Ours)	**64.13**	59.50	**76.50**	40.00	**76.50**	**67.00**	**53.50**	**64.13**	**64.13**	**62.00**	**63.07**
Qwen2.5-3B-Instruct											
GT_Funs	70.92	91.50	79.00	75.50	67.83	70.37	56.25	37.50	79.23	69.13	74.18
All_Funs	64.42	71.00	62.50	53.00	43.41	44.92	43.75	29.17	62.73	44.34	53.54
Top-K	64.42	74.50	71.00	45.00	41.09	48.53	**43.75**	29.17	63.73	46.71	55.22
HiTEC-ICL	64.17	71.50	64.00	55.50	44.19	45.68	**43.75**	37.50	63.79	45.23	54.51
ToolGT (Prompting)	58.67	71.50	54.50	52.50	**56.59**	55.37	**43.75**	25.00	59.29	54.92	57.11
Tool-DC (TF) (Ours)	**64.67**	**81.00**	**77.50**	**64.00**	48.84	**60.40**	**43.75**	**45.83**	**71.79**	**57.74**	**64.77**
Qwen2.5-7B-Instruct											
GT_Funs	72.25	96.00	91.00	84.00	77.13	78.73	62.50	66.67	85.81	78.02	81.92
All_Funs	65.33	82.00	85.00	70.50	58.14	63.91	75.00	54.17	75.71	62.77	69.24
Top-K	67.42	81.00	83.50	49.00	48.45	55.94	50.00	41.67	70.23	54.18	62.21
HiTEC-ICL	65.75	**86.00**	85.00	70.50	62.79	65.72	68.75	62.50	76.81	65.14	70.98
ToolGT (Prompting)	70.00	80.00	82.50	**71.50**	58.91	61.06	43.75	66.67	76.00	60.55	68.28
Tool-DC (TF) (Ours)	**84.40**	71.58	**94.75**	50.00	**91.00**	**91.00**	**84.00**	**84.40**	**84.40**	**70.00**	**77.20**

Table 8: Detailed results of different training-based methods on BFCL, which is a full version of Table 2.

Methods	Non-Live				Live				Overall		
	Simple	Multiple	Parallel	Parallel Multiple	Simple	Multiple	Parallel	Parallel Multiple	Non-Live	Live	Overall
Qwen2.5-7B-Instruct	75.33	94.50	91.50	84.50	76.74	74.93	62.50	70.83	86.46	67.44	76.95
-w Vanilla SFT	70.92	95.50	88.50	82.00	77.52	73.69	81.25	58.33	84.23	74.24	79.24
-w Tool-DC (TB)	71.17	95.00	93.00	87.50	83.72	79.58	56.25	54.17	86.67	79.64	**83.16**
Llama-3.2-3B-Instruct	70.67	92.50	88.50	79.00	58.33	65.12	57.64	25.00	82.67	58.33	70.50
-w Vanilla SFT	71.25	92.00	88.50	81.50	70.93	61.35	56.25	54.17	83.31	62.99	73.15
-w Tool-DC (TB)	71.67	89.50	89.50	82.00	68.60	64.01	18.75	41.67	83.17	63.95	**73.56**

Table 9: Detailed results of different training-free methods on ACEBench, which is a full version of Table 1. “Atom”, “Single Turn”, “Multiple Turn”, “Similar API”, and “Profile” are subsets of ACEBench.

Methods	Standard Setting						Extended Setting					
	Atom	Single Turn	Multiple Turn	Similar API	Profile	Overall	Atom	Single Turn	Multiple Turn	Similar API	Profile	Overall
Qwen2.5-1.5B-Instruct												
GT_Funs	56.00	38.50	28.00	52.00	54.00	47.92	56.00	38.50	28.00	52.00	54.00	47.92
All_Funs	56.67	35.00	29.00	50.00	38.00	46.33	27.67	15.00	5.00	32.00	26.00	22.00
Top-K	57.33	37.00	10.12	52.00	44.00	44.52	52.67	28.00	8.50	**40.00**	34.00	38.58
HiTEC-ICL	54.33	34.00	24.00	52.00	**48.00**	45.17	28.00	21.50	13.00	24.00	**44.00**	25.42
ToolGT (Prompting)	**60.67**	33.00	**30.00**	48.00	32.00	47.50	49.00	18.00	17.00	26.00	34.00	35.33
Tool-DC (TF)	56.33	**47.50**	23.00	**66.00**	44.00	**49.08**	**57.67**	**38.50**	**25.00**	**40.00**	40.00	**46.08**
Qwen2.5-3B-Instruct												
GT_Funs	63.33	47.50	28.00	60.00	62.00	54.42	63.33	47.50	28.00	60.00	62.00	54.42
All_Funs	57.33	45.50	17.00	50.00	48.00	47.25	43.00	32.00	17.00	48.00	34.00	36.50
Top-K	53.67	49.50	11.21	**62.00**	44.00	45.78	47.33	38.50	6.61	54.00	28.00	38.02
HiTEC-ICL	53.33	**52.00**	**30.00**	58.00	50.00	49.33	39.67	33.50	19.00	50.00	26.00	34.92
ToolGT (Prompting)	60.67	48.00	**30.00**	56.00	**54.00**	**52.50**	**56.67**	36.50	22.00	52.00	**50.00**	46.58
Tool-DC (TF)	**61.00**	40.50	23.00	50.00	50.00	49.42	55.67	**46.00**	**28.00**	**60.00**	36.00	**48.17**
Qwen2.5-7B-Instruct												
GT_Funs	71.00	61.00	52.00	68.00	62.00	65.17	71.00	61.00	52.00	68.00	62.00	65.17
All_Funs	62.33	63.50	45.00	64.00	64.00	59.92	65.33	54.50	42.00	62.00	56.00	58.58
Top-K	61.33	58.00	0.00	64.00	**62.00**	50.83	53.67	50.50	17.37	52.00	46.00	46.31
HiTEC-ICL	65.33	61.00	48.00	66.00	60.00	61.33	58.67	48.00	44.00	62.00	**58.00**	54.67
ToolGT (Prompting)	**79.67**	61.50	49.00	66.00	**62.00**	**68.92**	**71.33**	59.50	**53.00**	64.00	56.00	**64.42**
Tool-DC (TF)	68.00	**66.00**	**50.00**	**70.00**	58.00	64.00	60.00	**65.00**	48.00	**66.00**	54.00	58.83

Table 10: Detailed results of different training-based methods on Standard ACEBench, referred to Figure 5.

Methods	Atom	Single Turn	Multiple Turn	Similar API	Profile	Overall
Qwen2.5-7B-Instruct	62.33	63.50	45.00	64.00	64.00	59.92
-w Vanilla SFT	67.67	65.50	7.14	70.00	64.00	57.11
-w Tool-DC (TB)	76.33	47.00	22.94	74.00	64.00	**61.32**
Llama-3.2-3B-Instruct	33.33	21.00	8.00	34.00	50.00	28.50
-w Vanilla SFT	65.00	43.00	5.22	52.00	48.00	48.87
-w Tool-DC (TB)	72.67	43.00	12.71	56.00	38.00	**53.45**
Qwen3-4B-Instruct-2507	55.67	43.00	22.13	52.00	54.00	47.52
-w Vanilla SFT	75.00	65.50	16.67	70.00	66.00	62.53
-w Tool-DC (TB)	67.67	59.50	52.00	68.00	64.00	**63.42**