Improving Large Language Models Function Calling and Interpretability via Guided-Structured Templates

Hy Dang[1, 2*], Tianyi Liu[2], Zhuofeng Wu[2], Jingfeng Yang[2], Haoming Jiang[2], Tao Yang[2], Pei Chen[2], Zhengyang Wang[2], Helen Wang[2], Huasheng Li[2], Bing Yin[2], Meng Jiang[1, 2]

[1] University of Notre Dame, [2] Amazon

hdang@nd.edu

Abstract

Large language models (LLMs) have demonstrated strong reasoning and tool-use capabilities, yet they often fail in real-world tool-interactions due to incorrect parameterization, poor tool selection, or misinterpretation of user intent. These issues often stem from an incomplete understanding of user goals and inadequate comprehension of tool documentation. While Chain-of-Thought (CoT) prompting has proven effective for enhancing reasoning in general contexts, our analysis reveals that free-form CoT is insufficient and sometimes counterproductive for structured function-calling tasks. To address this, we introduce a curriculum-inspired framework that leverages structured reasoning templates to guide LLMs through more deliberate step-by-step instructions for generating function callings. Experimental results show that our method reduces tool-use errors, achieving 3–12% relative improvements over strong baselines across diverse model series and approaches. Moreover, our framework enhances the robustness, interpretability, and transparency of tool-using agents, advancing the development of more reliable AI assistants for real-world applications.

1 Introduction

Recent advancements in large language models (LLMs), including both closed- and open-source variants, have enabled a wide range of sophisticated capabilities, including complex reasoning, planning, and tool usage (Touvron et al., 2023; Liang et al., 2024), targeting develops comprehensive helpful agents (Guo et al., 2024a; Zhao et al., 2024; Zhu et al., 2025). For instance, LLM-based agents can now invoke external tools or APIs to fulfill user instructions, from simple tasks (e.g., checking a date) to complex workflows (e.g., booking hotels or making purchases) (Qu et al., 2025; Li, 2025). Despite these impressive abilities, current models frequently **fail to make correct function calls**—including errors such as incorrect parameterization, poor tool selection, or misinterpretation of user intent and hallucinations (Huang et al., 2023; Kokane et al.; Huang et al., 2025, 2024). Such failures directly impact the real-world applications, where functional correctness is critical for safety and trust (Xu et al., 2024; Zhang et al., 2024).

Additionally, many LLMs (Qin et al., 2023; Liu et al., 2024; Patil et al., 2024) generate function calls through a "black-box" process—providing no explanations for their selection of functions, choice of parameter values, or anticipated execution outcomes. This **lack of explainability** significantly hinders both systematic debugging and meaningful human oversight, a critical limitation that has been widely recognized across high-stakes domains such as healthcare, finance etc. (Barman et al., 2024; Ajwani et al., 2024; Zhu et al., 2024). Without transparent reasoning, stakeholders cannot effectively verify the suitability of tool usage, thereby increasing the risk of consequential errors.

Recent advances in prompting (Wei et al., 2022; Yao et al., 2023) have shown that LLMs benefit from intermediate reasoning steps, while subsequent research (Wu et al., 2024; Zhang and Ding, 2024; Feng et al., 2023; Chu et al., 2024) has emphasized the importance of task-specific supervision in effectively guiding models through complex solution space. These convergent findings substantiate our central hypothesis: **LLMs require structured, contextualized guidance—not merely generic heuristics—to achieve consistent and correct tool use.** This insight represents a fundamental shift from earlier approaches that relied primarily on general-purpose prompting strategies, suggesting instead that domain-adapted scaffolding is essential for reliable tool use.

In this work, we propose a **template-based reasoning framework** for function calling that structures the model's thought process according to

[*] Work done while the author was an intern at Amazon.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 24438–24454
November 4-9, 2025

task demands and tool specifications. Our template systematically guides models through critical sub-tasks in a manner aligned with human problem-solving patterns. Our initial experiments revealed that while fixed structured templates substantially improve execution accuracy and instruction following, they nonetheless exhibit persistent limitations in formatting adherence, logical consistency, and functional correctness. To address these challenges, we develop a pipeline (ToolGT) to construct synthetic finetuning dataset that systematically encodes reasoning patterns using structured templates. This dataset is purposefully designed to teach models in maintaining correct formatting conventions, executing and generating step-by-step analytical reasoning, and producing outputs that precisely align with API specifications, effectively targeting the specific weaknesses observed in Template-prompting approaches.

Extensive empirical results show that our Template-based prompting and training methods on our proposed structured template consistently outperform both No-Thought and CoT approaches across models and benchmarks. On average, compared to CoT-prompting, Template-prompting achieves improvements of +2.8/+1.7 and Template-based fine-tuning yields +1.0/+1.3 on BFCLv2 and Nexus over CoT-trained models, respectively.

In summary, our contributions are two-fold:

- **Template-Based Reasoning:** We develop an explicit prompting template that guide LLMs through critical stages of function calling, including tool understanding, parameter extraction, implicit conversion, and other task-specific requirements.

- **Structured Reasoning Dataset:** We introduce an approach for constructing the **G**uided-**T**emplate structured reasoning dataset (**ToolGT**) that effectively teaches models to improve accuracy and transparency across diverse tasks and model architectures.

We argue that equipping LLMs with curriculum-style reasoning templates offers a path toward more reliable and generalizable tool use. Rather than relying solely on unconstrained CoT reasoning, adaptive and context-specific structures can help models better align with user intent, execute accurate function calls, and provide interpretable justifications. This work establishes a foundation for future research in structured reasoning and advanced tool integration for next-generation LLM agents.

2 Methodology

We present a structured, template-based reasoning framework designed to enhance the function-calling capabilities of large language models (LLMs). Our approach comprises two main components: (1) prompting strategies (§2.2), and (2) fine-tuning strategies based on our proposed data construction method (§2.3).

2.1 Problem Definition

Given a user query x and a set of tools T, the function calling task traditionally aims to predict a function call y by modeling $p(y \mid x, T)$.

We extend this by introducing a structured reasoning chain r, yielding a joint modeling objective $p(r, y \mid x, T)$, where r provides interpretable, step-by-step justification for identifying, selecting, examining, and parameterizing functions. This improves transparency and reliability, especially in function calling domains.

To generate r, we aim to incorporate a guided reasoning template/curriculum c by two approaches: **prompting strategies** and **fine-tuning strategies**. In the prompting strategies, c is included in the prompt and the model will generate function call(s) based on the objective $p(r, y \mid x, T, c)$. In contrast, supervised fine-tuning follows a data construction pipeline (§2.3), which generates well-curated samples that follow c during training, and at inference the model predicts $p(r, y \mid x, T)$ without seeing c.

Further details on the template/curriculum and reasoning chain construction appear in §2.3.

2.2 Prompting Strategies

We first evaluate model's performance in following pre-defined set of curriculum by proposing a structured prompting methodology that guides LLMs through clearly defined reasoning steps when invoking external functions. Unlike naive CoT approaches, our method employs a structured template to enforce discrete reasoning stages.

Template-Based Curricula Prompting We formalize the reasoning process into a structured curriculum or template that the model must follow before invoking any function. Inspired by how human follow specific steps to compose a function calling, the template includes: (1) Identification of

functions, (2) Decision on relevancy of provided functions, (3) Examine relevant function documentation, (4) Extract and validate parameters provided by user queries, (5) Conversion of parameter types or implicit value if needed, (6) Draft a function, (7) Revalidate Function Call(s). The full detail of our template are represented in Appendix A.2 (**Detail**). By embedding this structured template into the prompt, we guide the model's reasoning process to follow specific curriculum to use tools.

2.3 Fine-Tuning Dataset Construction

As mentioned in previous section, we internalize structured reasoning capabilities within the model. Thus, we construct a high-quality **G**uided-**T**emplate dataset (**ToolGT**) for finetuning purposes. The construction process is illustrated in Figure 1 and consists of:

Initial Data Source We use a tool-oriented dataset, **ToolACE** (Liu et al., 2024), originally developed in multi-turn dialogues. Due to their conversational nature, these datasets often include multiple unrelated turns. To improve clarity and focus, we convert them into single-turn samples, each centered on a specific function-calling instance. Specifically, we use the ToolACE subset from HuggingFace[1], comprising 11,300 conversations.

After filtering and reformatting for BFCL-style function-calling, the dataset contains 11,488 single-turn samples. Following our data construction approach, we obtain **10,830 structured reasoning samples** for **ToolGT**. Each sample includes the components (T, x, r, y) from §2.1, and is generated using advanced LLMs guided by our structured template. All reasoning chains are then validated to ensure high data quality.

Generating Structured Reasoning Chains To generate high-quality reasoning sequences (r), we leverage advanced LLMs (e.g., GPT-4o-mini). Given the user query (x), the tool set (T), and the ground-truth function call (y), these models produce step-by-step reasoning chains explicitly guided by our template.

Function Calling Generation Using Generated Structured Reasoning Our goal is to obtain a good reasoning chain, thus each reasoning chain (r) is validated by feeding it back into the LLM alongside with (T, x) to obtain y' as $y' = \texttt{LLM}(x, T, r)$. And reasoning chain is kept if either method passes the training sample filtering process.

Training Sample Filtering To construct a high-quality fine-tuning dataset, we compare each generated function call (y'), generated using constructed reasoning chain r in previous step(s) to the ground truth (y) using a two-stage verification process:

1. **Manual Verification:** We apply Exact Match (EM) and Abstract Syntax Tree (AST) checks to verify syntactic and semantic equivalence. In more details, first, we compare generated with ground-truth function callings using EM approach. In the case that was marked as incorrect with EM, we figure out there are cases that the function-callings are still correct when orders of parameters are not in exact orders as in the ground truth function or there are optional parameter(s) passed but does not affect the execution. Thus, we perform additional steps of AST check to guarantee we cover these cases.

2. **LLM-based Verification:** Given the possibility of multiple valid outputs, we use an LLM—an increasingly evaluation tool (Zheng et al., 2023; Wang et al., 2025)—to judge if y' sufficiently answers the original query (x), even if it differs from y.

This filtering process ensures that the final fine-tuning dataset includes high-quality examples, covering valid responses—supporting robust reasoning and reliable function calling.

2.4 Finetuning Strategies

We train open-source models in the supervised fine-tuning (SFT) manner for our task, based on the constructed dataset described in §2.3. The models are initialized from publicly available pretrained checkpoints hosted on Hugging Face[2]. We experiment with several model families, as detailed in §3. Additional training details—including hyperparameters (see §A.2) and formatting templates (see §A.4)—are provided in Appendix A.

3 Experimental Setup

We conduct experiments to evaluate the effectiveness of explicitly embedding structured reasoning into large language models (LLMs) for enhancing their function-calling capabilities. Our evaluation

[1] https://huggingface.co/datasets/Team-ACE/ToolACE

[2] https://huggingface.co/

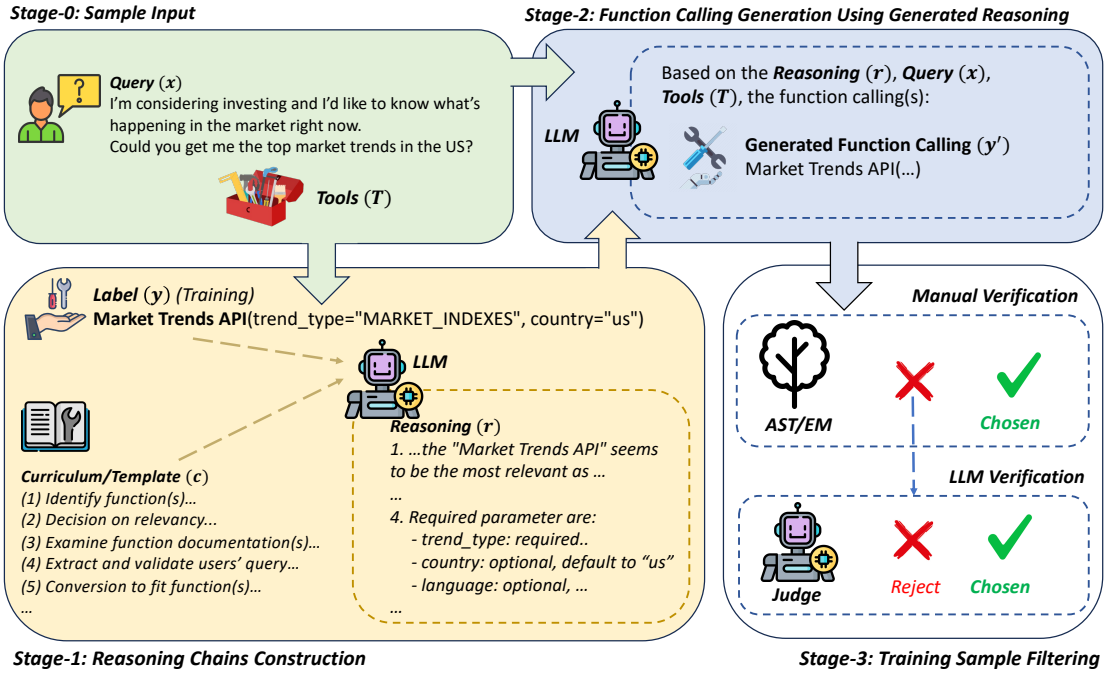

Figure 1: Overview of our supervised fine-tuning dataset construction **(ToolGT)** pipeline following three different stages. Stage 1 **(Reasoning Chains Construction)**—an LLM, guided by a curriculum/template c, generates a structured reasoning chain r based on query (x), set of available tools T and ground truth label y. Stage 2 **(Function Calling Generation Using Generated Reasoning)**—we evaluate the effectiveness of r by prompting an LLM to predict a function call y' conditioned on (x, T, r), without providing c at inference. Stage 3 **(Training Sample Filtering)**—to ensure high-quality supervision, we compare the predicted y' with the reference y using two rounds of verification: (1) Exact Match and AST-based structural comparison (AST/EM), and (2) LLM-based judgment to identify semantically equivalent alternatives. Only samples that pass verification are included in the final dataset.

focuses on two standard benchmarks: **BFCLv2**[3] and **Nexus** [4], which cover a wide range of function-calling scenarios, from simple queries to complex tasks for function calls.

It is important to note that we exclude executable test cases in the BFCLv2 benchmark from our analysis. These cases require real-time execution and external API calls, which are subject to strict rate limits. During preliminary evaluations, these constraints resulted in frequent execution errors and inconsistent results. To ensure a fair and stable comparison, we focus only on the non-executable subset of the benchmark.

3.1 Prompting-Based Experiments

We evaluate three prompting strategies: (1) **No Thought**, where models predict function calls without reasoning; (2) **Chain-of-Thought (CoT)**, which uses free-form reasoning ("Think-step-by-step") as guided curriculum; and (3) **Template** (ours), where models follow explicit, template-based reasoning prompts (see §2.2).

Our experiments include a diverse family of models, encompassing both closed- and open-source variants including:

- **Closed-source models** optimized for direct function calling include: **GPT-4o-FC** and **GPT-4o-mini-FC**, which embed templates without outputting intermediate steps. Meanwhile, **GPT-4o Prompting, GPT-4o-mini Prompting** support explicit reasoning.

- **Open-source** models evaluated with CoT and Template prompts/curricula come from diverse model sizes and families include: **LLaMA-3-70B-Instruct**, **LLaMA-3.1-8B-Instruct**, **Mistral-7B-Instruct-v0.3**, **Mistral-Nemo-12B**, and **Qwen-2.5-14B-Instruct**.

3.2 Supervised Fine-Tuning Experiments

To evaluate the structured reasoning on function-calling performance, we conduct supervised fine-tuning experiments using the dataset constructed

[3] https://gorilla.cs.berkeley.edu/blogs/12_bfcl_v2_live.html

[4] https://github.com/nexusflowai/NexusRaven-V2

following our pipeline described in §2.3. We compare three training strategies:

1. **No Thought** (Liu et al., 2024), where models were trained following ToolACE only on final function-calling;

2. **CoT-Training**, which added free-form reasoning "Think-step-by-step" following our data construction in §2.3;

3. **Template-Training** (ours), which used our structured reasoning template introduce in §2.2 with joint optimization over reasoning and function-calling.

Experiments were run on four open-source models across different model series: **LLaMA-3.1-8B-Instruct**, **Mistral-7B-Instruct-v0.3**, **Mistral-Nemo-12B-Instruct**, and **Qwen-2.5-14B-Instruct**.

3.3 Evaluation Metrics

As mentioned in the previous section, we conduct our experiments using **BFCL-v2** and **Nexus**, following the metrics outlined below.

BFCLv2 Benchmark : We organize BFCL-v2 evaluation into the following sub-categories:

1. **Non-Live Cases:** `simple`, `parallel`, `multiple`, `java`, `javascript`

2. **Live Cases:** `live_simple`, `live_multiple`, `live_parallel`, `live_parallel_multiple`

3. **Relevancy:** `relevance`, `live_relevance`, `live_irrelevance`, `irrelevance`

We compute a *Weighted Average* across task categories, accounting for task frequency and complexity. For BFCL averaging, we follow the official BFCL protocol[5], including `relevance` is grouped with Non-Live Categories, while `live_relevance` and `live_irrelevance` are grouped with Live.

Nexus Benchmark : We report both a *Weighted Average*, proportional to task frequency, and an *Unweighted Average*, computed as a simple mean over all tasks. These metrics capture performance across diverse task types and operational settings, reflecting both precision and robustness.

[5] gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html

4 Results and Analysis

Table 1: BFCL-v2 and Nexus performance using different prompting strategies for both close and open-sources models with different model families. Bold values indicate the best score per model with different strategies. Metrics are reported as Weighted / BFCL **(W / BFCL)** for BFCL-v2 and Weighted / Unweighted **(W / U)** for Nexus, following § 3.3. Higher means better ↑

Model	BFCL-v2 W / BFCL ↑	Nexus W / U ↑
Function Calling Generation Only		
GPT-4o-FC		
+ No Thought	73.78 / 74.79	– / –
+ CoT Prompting	77.40 / 78.83	– / –
+ Template Prompting	**78.99 / 80.26**	– / –
GPT-4o-mini-FC		
+ No Thought	73.52 / 74.72	– / –
+ CoT Prompting	77.34 / 78.71	– / –
+ Template Prompting	**78.30 / 79.79**	– / –
Function Calling and Thought Generation		
GPT-4o Prompting		
+ No Thought	79.40 / 78.52	47.83 / 45.42
+ CoT Prompting	79.29 / 78.70	47.11 / 44.87
+ Template Prompting	**81.21 / 80.83**	**53.18 / 51.51**
GPT-4o-mini Prompting		
+ No Thought	78.99 / 80.25	36.85 / 34.01
+ CoT Prompting	**79.87 / 80.59**	40.61 / 35.26
+ Template Prompting	79.70 / 80.24	**41.33 / 36.20**
LLaMA-3-70B-Instruct		
+ No Thought	70.89 / 71.50	44.51 / 35.76
+ CoT Prompting	75.75 / 75.05	46.10 / 38.46
+ Template Prompting	**77.78 / 76.81**	**47.40 / 41.85**
LLaMA-3.1-8B-Instruct		
+ No Thought	64.43 / 65.75	35.40 / 30.06
+ CoT Prompting	65.28 / 66.70	36.71 / 30.06
+ Template Prompting	**68.28 / 69.65**	**38.01 / 33.09**
Mistral-7B-Instruct-v0.3		
+ No Thought	**60.70 / 57.55**	10.84 / 9.05
+ CoT Prompting	51.11 / 48.94	**26.01 / 22.26**
+ Template Prompting	48.83 / 46.88	25.29 / 20.80
Mistral-Nemo-12B		
+ No Thought	57.92 / 58.90	25.72 / 22.08
+ CoT Prompting	63.31 / 65.19	31.50 / 27.78
+ Template Prompting	**64.71 / 66.32**	**32.95 / 28.46**
Qwen-2.5-14B-Instruct		
+ No Thought	**76.22 / 77.33**	43.21 / 38.57
+ CoT Prompting	68.06 / 62.25	41.33 / 35.38
+ Template Prompting	74.37 / 73.28	**44.07 / 40.78**

We begin by examining the impact of Template-based reasoning through prompting strategies (see

§4.1), followed by fine-tuning strategies (see §4.2).

Throughout this section, we report results using the *BFCL average* for BFCLv2 and the *unweighted average* for Nexus to ensure consistency and clarity. Full metric results are shown in Table 1 and Table 2. Additional sub-category results for both benchmarks are provided in Appendix A.3.

4.1 Impact of Structured Prompting

Structured Reasoning via Template Prompting Outperforms Free-form CoT First, we examine *whether providing a well-structured template/curriculum in the prompt can improve models' function-calling capabilities compared to no-thought or a naive CoT guidance approach*. We present the summary results in Table 1.

Across both the BFCL-v2 and Nexus benchmarks, our results suggest that **Template Prompting** often yields better performance than CoT and No Thought, particularly across several model types and evaluation metrics on BFCL-v2 and Nexus, with further discussion on exceptions across model families provided later in this section. For example, in the function-calling generation setting, **GPT-4o-FC** achieves its best performance with Template Prompting (80.26), outperforming CoT (78.83) and No Thought (74.79). **GPT-4o-mini-FC** follows a similar trend, reaching 79.79 compared to 78.71 (CoT) and 74.72 (No Thought).

This pattern additionally holds across open-source models. Large Open-Source model **LLaMA-3-70B-Instruct** outperforms with our Template Prompting approach on both BFCLv2 and Nexus (77.78 / 47.40), outperform other counterparts, including CoT (75.75 / 46.10) and No Thought (70.89 / 35.76).

Similarly, smaller open-source model, **LLaMA-3.1-8B-Instruct** shows notable gains with Template Prompting on both BFCL (68.28) and Nexus (33.09), outperforming CoT (66.70 / 30.06) and No Thought (65.75 / 30.06), suggesting that structured prompting more effectively guides models in tool usage. Results from **Mistral-Nemo-12B** also show benefits, with improvements of +7.42 and +1.13 on BFCL, and +6.38 / +0.68 on Nexus, compared to No Thought and CoT, respectively. These results underscore the value of structured templates in improving model's performance.

Template Prompting Maintains Interpretability Without Sacrificing Performance A noteworthy observation comes from **Qwen-2.5-14B-Instruct**, where adding reasoning steps—either through CoT or Template Prompting—lowers performance on the BFCL-v2 benchmark compared to the No Thought. We hypothesize that this is due to the model being heavily trained on direct function-calling tasks, that introducing reasoning may interfere with its learned execution patterns.

However, Template Prompting (73.28) still performs much closer to the No Thought baseline (77.33) than CoT does (62.25), highlighting the greater robustness of structured templates over free-form reasoning. Moreover, on the Nexus benchmark, Template Prompting achieves the highest score (40.78), representing relative gains of +5.7% over No Thought (38.57) and +15.26% over CoT (35.38). These results demonstrate that Template Prompting preserves interpretability through intermediate reasoning without compromising performance, even in high-capacity models.

Inconsistencies in Smaller Variants Due to Formatting and Instruction-Following Limitations. An exception to the overall trend is **Mistral-7B-Instruct-v0.3**, which performs worse with Template-based prompting on the BFCLv2 (46.88) compared to both CoT prompting (48.94) and No Thought (57.55). Analysis of the model's outputs suggests that this underperformance stems from its difficulty in following structured reasoning templates and formatting issues—due to limited instruction tuning or insufficient exposure to such formats during pretraining. A similar pattern is observed on the Nexus benchmark, where CoT prompting outperforms Template prompting (22.26 vs. 20.80), further emphasizing the need for targeted fine-tuning to enable smaller models to effectively leverage structured prompting strategies.

4.2 Impact of Internalizing Structured Reasoning via Fine-Tuning

As mentioned in previous section, we aim to resolve formatting and instruction-following limitations introduced by some model(s) and further enhance the model(s) performance.

Therefore, in this section, we evaluate *whether internalizing structured-reasoning models via fine-tuning can improve the model performance and resolve formatting and instruction-following issue(s).* The summary results on all benchmark with experiments across different models' series is included in Table 2.

Model	BFCL-v2 W / BFCL ↑	Nexus W / U ↑
LLaMA-3.1-8B-Instruct		
+ No Thought-Training	69.73 / 72.16	35.98 / 29.82
+ CoT-Training	73.33 / 74.74	**36.85** / 30.19
+ **Template-Training**	**74.10 / 75.28**	36.27 / **30.23**
Mistral-7B-Instruct-v0.3		
+ No Thought-Training	64.65 / 66.51	26.59 / 21.78
+ CoT-Training	67.06 / 69.16	24.85 / 20.55
+ **Template-Training**	**69.87 / 71.40**	**27.02 / 23.17**
Mistral-Nemo-12B-Instruct		
+No Thought-Training	74.87 / 76.44	32.66 / 26.46
+CoT-Training	76.60 / 77.72	35.69 / 31.00
+ **Template-Training**	**77.40 / 78.63**	**37.28 / 32.46**
Qwen-2.5-14B-Instruct		
+ No Thought-Training	77.06 / 78.61	40.74 / 35.72
+ CoT-Training	77.92 / 79.46	40.75 / 35.85
+ **Template-Training**	**78.55 / 79.83**	**42.20 / 37.05**

Table 2: Performance on BFCL-v2 and Nexus using fine-tuned models trained with **ToolACE**, **CoT**, or **Template-constructed (ToolGT)**. Bold values indicate the best score for each model across both benchmarks. We report the Weighted / BFCL **(W / BFCL)** average metric for BFCL-v2, and the Weighted / Unweighted **(W / BFCL)** average metric for Nexus, following the evaluation protocol described in § 3.3. Higher means better ↑

Template-Based Fine-Tuning Consistently Enhances Performance and Robustness According to our results, template-based fine-tuning (Template-Training) consistently yields the highest performance across both benchmarks for all evaluated open-source models.

LLaMA-3.1-8B-Ins improves to 75.28 with Template-Training, surpassing CoT (74.74) and the No Thought-Training baseline (72.16). **Mistral-Nemo-12B-Ins** similarly achieves its best result with Template-Training (78.63), outperforming CoT-Training (77.72) and No Thought-Training (76.44). These gains highlight the effectiveness of explicitly internalizing structured reasoning through fine-tuning. Additionally, Template-Training provides consistent guidance for function usage, improving both accuracy and interpretability in function-calling tasks compared to CoT-Training approach.

Training Enables Models to Effectively Utilize Structured Reasoning Mistral-7B-Instruct-v0.3, which previously underperformed with Template-prompting (§ 4.1), struggling to follow structured formats and falling below even the No Thought prompting. After Template-based fine-tuning, however, the model shows notable gains—achieving 71.40 on BFCL (vs. 66.51 for No Thought and 69.16 for CoT) and 23.17 on Nexus (vs. 21.78 for No Thought). Interestingly, fine-tuning with CoT samples results in lower Nexus performance (20.55), suggesting that unguided reasoning may hinder generalization.

These results support our earlier hypothesis: reasoning formats require dedicated training. While prompting alone may be insufficient—especially for smaller models—supervised fine-tuning enables systematic reasoning, effectively bridging the gap between structure and performance.

Broad Applicability Across Different Models Template-based training proves effective across model series. For example, **Qwen-2.5-14B-Ins** achieves top performance across both benchmarks (BFCL: 79.83; Nexus: 37.05). Notably, CoT fine-tuning yields only marginal gains over baseline in the Nexus benchmark (e.g., 35.85 vs. 35.72, respectively), whereas Template-training provides meaningful improvements (37.05), account for approximately +3.5% relative improvements over both No Thought-Training and CoT-Training.

These results suggest that template supervision generalizes well across models, offering a robust and interpretable foundation for enhancing tool use, particularly in complex function-calling tasks.

5 Ablation Study

5.1 Impact of Template Complexity

To study the effectiveness of our proposed template **Detail**. We experiment with different types of templates (**Simple, Claude, Detail**) (See §A.2 for details) with results summarized in Table 3.

According to Table 3, the **Detail** template consistently achieves the highest overall accuracy. For example, it outperforms other variants across most tasks (LLaMA-3.1-8B-Instruct: 74.10 vs. 71.27 for **Simple**, 70.20 for **Claude** in overall accuracy). Additionally, we make interesting observation related to the benefit(s) of different type of template. In more details, **Simple** occasionally performs better on specific subtasks, such as *Relevancy*, suggesting that task-specific template complexity may be beneficial.

These results highlight a trade-off between **specificity** and **simplicity**, motivating future research on **adaptive strategy** that dynamically adjust reason-

ing depth based on task characteristics, which we leave it for future work.

Table 3: Performance across different templates used for training data generation with Mistral-v0.3 and Llama-3.1 models. "No Thought" denotes training directly on function calls, whereas "With Thought" explicitly includes structured reasoning. Evaluations are based on BFCL benchmark using *Weighted average*. Higher means better ↑

Category	Simple	Claude	Detail
Mistral-v0.3-7B-Ins			
No Thought Training			
Overall Score	66.44	47.73	64.65
With Thought Training			
Overall Score	68.63	60.64	**69.87**
Non-Live Average	78.52	70.26	**81.22**
Live Average	55.35	47.19	**59.92**
Relevancy	**74.13**	66.59	70.07
Llama-3.1-8B-Ins			
No Thought Training			
Overall Score	64.49	61.55	62.76
With Thought Training			
Overall Score	71.27	70.20	**74.10**
Non-Live Average	80.43	79.57	**84.61**
Live Average	55.58	54.60	**63.68**
Relevancy	**80.28**	78.90	75.61

5.2 Lack of Training Samples Can Harm in Complex Scenarios.

During our analysis, we observed unexpected performance degradation in **LLaMA-3.1-8B-Instruct** and **Qwen-2.5-14B-Instruct** after fine-tuning on ToolACE (No Thought), CoT-Training (CoT), and ToolGT (Template), across all reasoning strategies (see Table 1 and Table 2) in **Nexus**. For example, in LLaMA, No Thought dropped slightly from 30.06 to 29.82, while Template prompting decreased more noticeably from 33.09 to 30.23. The drop was more pronounced in Qwen, with No Thought falling from 38.57 to 35.72—a relative decrease of about 7%. We hypothesize that this degradation is due to ToolACE's limited coverage of complex, nested function-call scenarios, a limitation also present in the BFCLv2 evaluation.

To investigate, we analyzed performance on a *non-nested* subset of Nexus (Figure 2, Figure 3). In this setting, fine-tuned models improved consistently across all prompting strategies. For instance, LLaMA's Template-trained variant achieved the highest score (43.67), and Qwen showed gains in the first two prompting conditions. However, Qwen's Template-trained model, while outperforming other training strategies, still lagged behind zero-shot template prompting (54.22 vs. 56.17).

These results suggest that without adequate training coverage of compositional tool use, fine-tuned models may overfit to simpler patterns, impairing performance on more complex tasks. We leave the development of datasets targeting nested and compositional reasoning for future work.

6 Related Work

Research on methods that enable tool usage in language models broadly falls under two main categories: prompting-based (§6.1) and tuning-based (§6.2) approaches, with some recent attention to extended reasoning techniques (§6.3).

6.1 Prompting-Based Methods

Prompting methods equip LLMs with textual tool descriptions to help them select and invoke tools for tasks such as question answering (Yao et al., 2023; Lu et al., 2023) and mathematical problem solving (Imani et al., 2023; Das et al., 2024). More advanced prompting strategies introduce step-by-step reasoning or iterative interactions (Paranjape et al., 2023; Wei et al., 2022; Sun et al., 2023). However, these approaches often rely on simplistic tool definitions and require multiple generation rounds, limiting efficiency and scalability. In contrast, our work focuses on enabling accurate tool use through structured, template-guided prompting while maintaining efficiency.

6.2 Tuning-Based Methods

Recent work has explored fine-tuning models for function calling (Gao et al., 2024; Schick et al., 2023; Qin et al., 2024). For example, Toolformer (Schick et al., 2023) uses specialized tokens to guide tool use, while ToolkenGPT (Hao et al., 2023) encodes tools as tokens to generalize to unseen tools. A similar work, ToolGen (Wang et al.), also aims to embed tools as tokens and tries to unify retrieval and tool-call generation. Other methods train on curated synthetic datasets from different approaches (Guo et al., 2024b; Liu et al., 2024; Li et al., 2023); for example, ToolACE (Liu et al., 2024) uses a multi-agent framework, while Gorilla builds on APIBench (Patil et al., 2024)

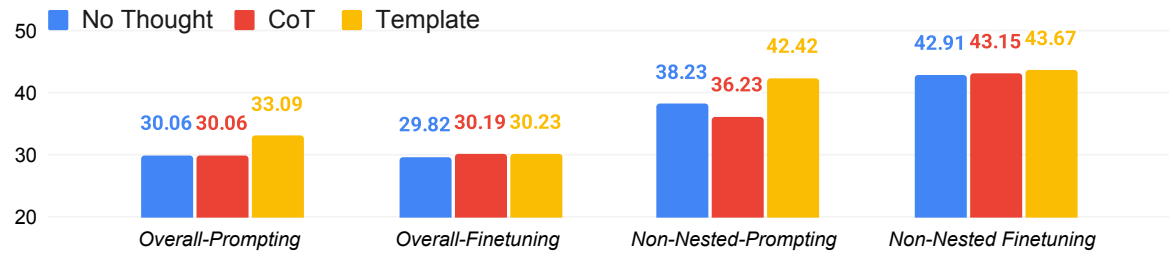

Figure 2: Performance comparison of LLaMA-3.1-8B-Instruct across prompting strategies (No Thought, CoT, Template) before and after fine-tuning (SFT) on Nexus benchmark. The results highlight performance degradation in overall scenarios (Left 2 groups), contrasted by clear improvements on the non-nested subset (Right two groups)

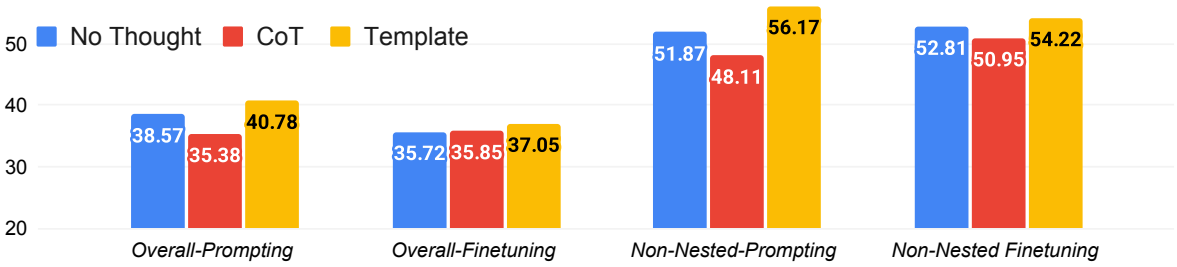

Figure 3: Performance comparison of Qwen-2.5-14B-Instruct across prompting strategies (No Thought, CoT, Template) before and after fine-tuning (SFT) on Nexus benchmark. The results highlight performance degradation in overall scenarios (Left 2 groups), contrasted by clear improvements on the non-nested subset (Right two groups)

to support diverse APIs. While these approaches improve accuracy, they primarily map queries directly to function calls without intermediate reasoning. TRICE (Qiao et al., 2024) introduces post-execution feedback; in contrast, our approach emphasizes pre-execution structured reasoning to enhance interpretability and performance. Additionally, there is research that aims to provide some form(s) of template reasoning, such as Synthesize Step-by-Step (Li et al., 2024), which employs templates to train an LLM-based data generator that produces questions and step-by-step rationales for chart visual question answering problems. In contrast, our work targets function calling and tool-use reasoning, aiming to enhance execution correctness and interpretability during inference.

6.3 Extended CoT for Long Reasoning

Recent models such as Deepseek-R1 (Guo et al., 2025) adopt extended CoT reasoning to improve interpretability and robustness. However, these approaches often introduce significant token overhead, length constraints, and formatting issues, limiting their practicality in latency-sensitive applications, especially in small models (Feng et al., 2025; Guo et al., 2025; Dang and Ngo, 2025). Additionally, this reinforcement learning techniques requires intensive training architectures (Parmar and Govindarajulu, 2025). Thus, we propose a complementary, template-driven strategy dataset that could be used to train models to perform structured, grounded reasoning. This method maintains interpretability while reducing error rates and improving efficiency.

7 Conclusion

In this paper, we propose a structured, template-based approach to enhance the function-calling capabilities of LLMs. By following carefully designed templates tailored for function call generation, our method guides models through deliberate, step-by-step reasoning rather than relying on naive, unguided outputs. Experimental results demonstrate that both structured prompting and supervised fine-tuning significantly improve function call accuracy while also enhancing interpretability across various models and benchmarks. Our contributions include explicit reasoning templates for function calling, a synthetic dataset construction method, and empirical evidence of performance gains. Future directions include extending structured reasoning to broader decision-making tasks and adaptive curricula to further improve the reliability and transparency of LLM-based agents.

Limitations

Our study has several limitations that present opportunities for future research. First, we only consider a subset of ToolACE, as only a limited portion of the dataset has been publicly released. This restricts the comprehensiveness of our evaluations and limits the potential of ToolACE in data construction. In future work, we plan to augment these cases with additional function-calling datasets—including more complex, nested scenarios that ToolACE lacks—to construct a more rigorous and diverse training corpus under our proposed framework. Second, our analysis reveals that different prompting templates can offer advantages in specific function-calling scenarios. This suggests a promising direction for developing models that can adaptively select and apply the most suitable template based on contexts and scenarios. Finally, our current experiments are limited to single-turn function-calling tasks. Currently, we do not evaluate or develop our framework to multi-turn scenarios, which introduced in BFCLv3 [6]. Thus, future work should incorporate multi-turn tool, function calling settings.

Acknowledgements

This work was supported by an internship at Amazon and NSF IIS-2119531, IIS-2137396, IIS-2142827, and IIS-2234058. We also appreciate the support from the Foundation Models and Applications Lab of Lucy Institute and ND-IBM Tech Ethics Lab.

References

Rohan Ajwani, Shashidhar Reddy Javaji, Frank Rudzicz, and Zining Zhu. 2024. Llm-generated black-box explanations can be adversarially helpful. *arXiv preprint arXiv:2405.06800*.

Kristian Gonzalez Barman, Nathan Wood, and Pawel Pawlowski. 2024. Beyond transparency and explainability: on the need for adequate and contextualized user guidelines for llm use. *Ethics and Information Technology*, 26(3):47.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1173–1203, Bangkok, Thailand. Association for Computational Linguistics.

Quy-Anh Dang and Chris Ngo. 2025. Reinforcement learning for reasoning in small llms: What works and what doesn't. *arXiv preprint arXiv:2503.16219*.

Debrup Das, Debopriyo Banerjee, Somak Aditya, and Ashish Kulkarni. 2024. MATHSENSEI: A tool-augmented large language model for mathematical reasoning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 942–966, Mexico City, Mexico. Association for Computational Linguistics.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36:70757–70798.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024a. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024b. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156, Bangkok, Thailand. Association for Computational Linguistics.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36:45870–45894.

Xiaowei Huang, Wenjie Ruan, Wei Huang, Gaojie Jin, Yi Dong, Changshun Wu, Saddek Bensalem,

[6] https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html

Ronghui Mu, Yi Qi, Xingyu Zhao, and 1 others. 2024. A survey of safety and trustworthiness of large language models through the lens of verification and validation. *Artificial Intelligence Review*, 57(7):175.

Yue Huang, Chujie Gao, Siyuan Wu, Haoran Wang, Xiangqi Wang, Yujun Zhou, Yanbo Wang, Jiayi Ye, Jiawen Shi, Qihui Zhang, and 1 others. 2025. On the trustworthiness of generative foundation models: Guideline, assessment, and perspective. *arXiv preprint arXiv:2502.14296*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2023. Metatool benchmark: Deciding whether to use tools and which to use. *arXiv preprint arXiv: 2310.03128*.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. MathPrompter: Mathematical reasoning using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 37–42, Toronto, Canada. Association for Computational Linguistics.

Shirley Kokane, Ming Zhu, Tulika Manoj Awalgaonkar, Jianguo Zhang, Akshara Prabhakar, Thai Quoc Hoang, Zuxin Liu, Rithesh RN, Liangwei Yang, Weiran Yao, and 1 others. Toolscan: A benchmark for characterizing errors in tool-use llms. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

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

Zhuowan Li, Bhavan Jasani, Peng Tang, and Shabnam Ghadar. 2024. Synthesize step-by-step: Tools templates and llms as data generators for reasoning-based chart vqa. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13613–13623.

Zijing Liang, Yanjie Xu, Yifan Hong, Penghui Shang, Qi Wang, Qiang Fu, and Ke Liu. 2024. A survey of multimodel large language models. In *Proceedings of the 3rd International Conference on Computer, Artificial Intelligence and Control Engineering*, pages 405–409.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36:43447–43478.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

Manojkumar Parmar and Yuvaraj Govindarajulu. 2025. Challenges in ensuring ai safety in deepseek-r1 models: The shortcomings of reinforcement learning strategies. *arXiv preprint arXiv:2501.17030*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. 2024. Making language models better tool learners with execution feedback. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3550–3568, Mexico City, Mexico. Association for Computational Linguistics.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, and 1 others. 2024. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *CoRR*, abs/2307.16789.

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.

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

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation, 2024a. *URL https://arxiv. org/abs/2410.03439*.

Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. 2025. Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering. *arXiv preprint arXiv:2502.06193*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhuofeng Wu, Richard He Bai, Aonan Zhang, Jiatao Gu, V.G.Vinod Vydiswaran, Navdeep Jaitly, and Yizhe Zhang. 2024. Divide-or-conquer? which part should you distill your LLM? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2572–2585, Miami, Florida, USA. Association for Computational Linguistics.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. 2024. Reducing tool hallucination via reliability alignment. *arXiv preprint arXiv:2412.04141*.

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

Xiang Zhang and Dujian Ding. 2024. Supervised chain of thought. *arXiv preprint arXiv:2410.14198*.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. 2024. ToolBeHonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11388–11422, Miami, Florida, USA. Association for Computational Linguistics.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. 2025. KnowAgent: Knowledge-augmented planning for LLM-based agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3709–3732, Albuquerque, New Mexico. Association for Computational Linguistics.

Zining Zhu, Hanjie Chen, Xi Ye, Qing Lyu, Chenhao Tan, Ana Marasović, and Sarah Wiegreffe. 2024. Explanation in the era of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 5: Tutorial Abstracts)*, pages 19–25.

A Appendix

A.1 Training & Hyperparameters

Table 4 provides the detailed hyperparameters used for supervised fine-tuning experiments. Training is conducted on 8 NVIDIA A100 80GB GPUs, requiring approximately 15 GPU hours for each finetuning experiment.

Table 4: Training hyperparameters for supervised fine-tuning experiments.

Hyperparameter	Value
Batch Size (per device)	2
Thought Generation Model	GPT-4o-mini
Training Sample Verification	GPT-4o-mini
Learning Rate	2e-5
LR Scheduler	cosine
Warmup Ratio	0.1
Epochs	3
Max Sequence Length	8192

A.2 Templates Used For Data Construction

Table 5 summarizes the templates (Claude, Simple, Detail) used in our training data construction and experiments, along with the number of helpful training samples generated using GPT-4o-mini. Among these, the **Detail** template serves as the

primary design for both our prompting and data construction strategies.

As discussed in §5.1, we observe varying effectiveness across templates, with the **Detail** template consistently outperforming the others. For prompting strategies, we also adopt **Detail** in our Template-Prompting experiments.

Template	Prompt
Detail (10,830)	1.Identify which function or set of functions best fit the user query based on function descriptions. 2.Pick that function or set of functions to fulfill the user's request. 3. After selecting the function(s), carefully examine the function documentation. 4. Analyze the provided parameters, considering descriptions, parameter types, and optionality. 5.Extract relevant information from user queries, performing necessary type conversions. 6. Draft the function call(s) fulfilling the user's request. 7. Revalidate the composed functions, ensuring they satisfy both documentation and the user's query.
Claude[7] **(9,307)**	First, determine the most relevant tool provided to answer the user's request. Then, review each required parameter for the selected tool, verifying if the user explicitly provided or implicitly offered sufficient information for inference. If all required parameters can be directly or reasonably inferred, proceed to invoke the function. If a required parameter is missing, do not invoke the function and instead request the missing parameters from the user. Avoid requesting optional parameters if not explicitly provided.
Simple (10,300)	First, identify the appropriate tool(s) for answering the user's request. Then analyze the query to formulate the necessary function call parameters. If no suitable tools are available or insufficient information is provided, refrain from making a function call.

Table 5: Template-wise statistics of helpful training samples used during dataset creation with structured prompts.

A.3 Detailed Experiment Results

BFCL-v2 Evaluation Details For this section, we present the full BFCLv2 results of our experiments including both prompting and finetuning strategies in Table 6.

Nexus Evaluation Details We report the full Nexus results from our experiments in Table 7, covering both prompting and fine-tuning strategies across eight dataset sub-categories[8]. Sub-category analysis reveals interesting patterns: while our template-based prompting consistently outperforms baselines in many categories and achieves the best overall performance, we also observe cases where other approaches—such as direct (No-Thought) generation and naive CoT prompting—prove beneficial. We hypothesize that this may be due to prior model exposure to specific functions or APIs, varying task complexity, user query, or differences in documentation structure. These findings further underscore the promise of **adaptive template** strategies and point to the need for more comprehensive and fair evaluation protocols, which we leave for future work.

A.4 Prompts For Data Construction

We provide prompts for each stage of the **ToolGT** construction process, including: reasoning chain generation (Table 8), function-call generation using the generated reasoning (Table 9), and training sample filtering (Table 10). Finally, we combine these steps to produce a training example used for fine-tuning, shown in Table 11.

[7] https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview#chain-of-thought-tool-use

[8] https://huggingface.co/spaces/Nexusflow/Nexus_Function_Calling_Leaderboard

Table 6: Sub-category results on BFCL-v2 using prompting and fine-tuning strategies. Bold values indicate the best performance per model. Averages of each categories computed using the weighted metric. Meanwhile, **W / BFCL** average metric still follows evaluation protocol mentioned in §3.3

Model	Non-Live Cases	Live-Cases	Relevancy	W / BFCL ↑
Prompting Strategies				
GPT-4o-FC				
+No Thought	84.87	69.81	67.32	73.78 / 74.79
+CoT	86.00	63.45	84.95	77.40 / 78.83
+Template	87.65	66.51	84.78	**78.99 / 80.26**
GPT-4o-mini-FC				
+No Thought	85.22	69.36	66.70	73.52 / 74.72
+CoT	84.52	64.57	84.95	77.34 / 78.71
+Template	86.78	67.57	82.27	**78.30 / 79.79**
GPT-4o-mini-Prompting				
+No Thought	88.78	73.71	75.34	78.99 / 80.25
+CoT	85.91	72.58	82.26	**79.87 / 80.59**
+Template	86.43	71.99	81.92	79.70 / 80.24
LLaMA-3.1-8B-Instruct				
+No Thought	86.35	66.37	40.40	64.43 / 65.75
+CoT	81.91	62.25	52.25	65.28 / 66.70
+Template	84.96	65.10	55.36	**68.28 / 69.65**
Mistral-7B-Instruct-v0.3				
+No Thought	64.17	54.60	64.27	**60.70 / 57.55**
+CoT	56.61	54.75	41.43	51.11 / 48.94
+Template	57.57	51.01	37.63	48.83 / 46.88
Mistral-Nemo-12B-Instruct				
+No Thought	86.17	72.96	12.46	57.92 / 58.90
+CoT	84.09	66.59	38.84	63.31 / 65.19
+Template	83.91	68.09	41.70	**64.71 / 66.32**
Qwen-2.5-14B-Instruct				
+No Thought	88.96	70.04	70.68	**76.22 / 77.33**
+CoT	61.57	70.26	71.97	68.06 / 62.25
+Template	79.22	71.38	73.01	74.37 / 73.28
Finetuning Strategies				
LLaMA-3.1-8B-Instruct				
+No Thought	87.22	63.22	59.86	69.73 / 72.16
+CoT	82.70	63.14	75.78	73.33 / 74.74
+Template	84.61	63.68	75.61	**74.10 / 75.28**
Mistral-7B-Instruct-v0.3				
+No Thought	83.39	64.27	46.45	64.65 / 66.51
+CoT	79.39	59.02	64.10	67.06 / 69.16
+Template	81.22	59.92	70.07	**69.87 / 71.40**
Mistral-Nemo-12B-Instruct				
+No Thought	85.04	63.82	77.51	74.87 / 76.44
+CoT	84.52	68.92	77.60	76.60 / 77.72
+Template	86.78	68.54	78.31	**77.40 / 78.63**
Qwen-2.5-14B-Instruct				
+No Thought	88.09	67.19	77.51	77.06 / 78.61
+CoT	87.04	68.83	79.32	77.92 / 79.46
+Template	88.70	69.22	79.24	**78.55 / 79.83**

Table 7: Detailed results on sub-categories of the Nexus Function Calling benchmark using prompting and fine-tuned strategies on open-source models. **VT** refers to the VirusTotal API, while **VT (N)** and **VT (P)** represent nested and parallel cases, respectively. Bold values indicate the best for each model within each setting. Averages are computed using the weighted/unweighted **(W/U) mean** in §3.3.

Model	NVDLibrary	VT	Places	Climate	OTX	VT (N)	VT (P)	CVECPE	W/U ↑
Prompting Strategies									
LLaMA-3.1-8B-Instruct									
+No Thought	38.46	68.87	16.67	9.64	82.61	8.16	14.29	1.79	35.40 / 30.06
+CoT	50.00	66.89	8.33	12.18	84.78	8.16	4.76	5.36	36.71 / 30.06
+Template	43.59	68.87	18.75	13.20	83.70	12.24	19.05	5.36	**38.01 / 33.09**
Mistral-7B-Instruct-v0.3									
+No Thought	24.35	12.58	4.17	7.11	20.65	0.00	0.00	3.57	10.84 / 9.05
+CoT	35.90	48.34	8.33	5.58	78.26	0.00	0.00	1.79	**26.01 / 22.26**
+Template	41.03	44.37	8.33	6.09	63.04	0.00	0.00	3.57	25.29 / 20.80
Mistral-Nemo-12B-Instruct									
+No Thought	41.03	37.09	8.33	6.09	79.35	0.00	4.76	0.00	25.72 / 22.08
+CoT	42.31	56.29	10.42	5.58	84.78	2.04	19.05	1.79	31.50 / 27.78
+Template	50.00	58.94	14.58	7.61	80.43	0.00	14.29	1.79	**32.95 / 28.46**
Qwen-2.5-14B-Instruct									
+No Thought	60.25	79.47	29.17	12.69	90.22	8.16	28.57	0.00	43.21 / 38.57
+CoT	73.07	78.14	10.42	9.14	89.13	4.08	19.05	0.00	41.33 / 35.38
+Template	79.49	76.16	25.00	10.15	89.13	4.08	33.33	8.93	**44.07 / 40.78**
Finetuning Strategies									
LLaMA-3.1-8B-Instruct									
+No Thought	46.15	69.54	36.36	85.00	88.04	0.00	18.18	0.00	35.98 / 29.82
+CoT	62.82	70.86	45.45	90.00	76.09	0.00	0.00	0.00	**36.85** / 30.19
+Template	66.67	60.93	40.91	95.00	85.87	0.00	0.00	0.00	36.27 / **30.23**
Mistral-7B-Instruct-v0.3									
+No Thought	37.18	43.71	6.25	5.58	81.52	0.00	0.00	0.00	26.59 / 21.78
+CoT	37.18	43.05	12.50	6.59	63.04	2.04	0.00	0.00	24.85 / 20.55
+Template	43.59	40.40	14.58	8.12	73.91	0.00	4.76	0.00	**27.02 / 23.17**
Mistral-Nemo-12B-Instruct									
+No Thought	34.62	66.89	10.42	7.61	82.61	0.00	9.52	0.00	32.66 / 26.46
+CoT	65.38	60.93	20.83	7.61	83.70	0.00	9.52	0.00	35.69 / 31.00
+Template	64.10	64.90	20.83	8.63	86.96	0.00	14.29	0.00	**37.28 / 32.46**
Qwen-2.5-14B-Instruct									
+No Thought	71.79	72.84	25.00	10.15	86.92	0.00	19.05	0.00	40.74 / 35.72
+CoT	82.05	74.83	22.92	8.63	79.35	0.00	19.05	0.00	40.75 / 35.85
+Template	75.64	76.82	27.08	10.15	85.87	0.00	19.05	1.79	**42.20 / 37.05**

Stage-1: Reasoning Chains Construction	
Role	**Content**
System	You are an expert in composing functions. You are given a question, a set of possible functions and the ground truth function call(s). Based on the question and the ground truth function call(s), you will need to generate the analysis and thought following the given curriculum steps by steps, however, you must pretend that you do not know the ground truth information and assumptions. 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. Here is a list of functions in JSON format that you can invoke. {FUNCTIONS HERE} When composing your analysis, you SHOULD follow this curriculum to have a correct function calling and put your analysis followed this curriculum in <THINKING></THINKING> tags. {GUIDED-TEMPLATE HERE} The output format of all user requests are: <THINKING>[Put your thought based on the curriculum step by step here]</THINKING>
User	User request: {user request} Ground truth function calling(s):{GROUND TRUTH}

Table 8: **Stage-1 Prompt** during our data construction process, which provides ground truth, template and user query and asking models to generate analysis based on provided information. Note that in this step, we do not allow the models to explicitly say that they know the answer during reasoning generation.

Stage-2: Function-call Generation Using the Generated Reasoning Prompt	
Role	**Content**
System	You are an expert in composing functions. You are given a question, a set of possible functions and an analysis to come up with correct function calling(s). Based on the question and provided thinking process, you will need to make one or more function/tool calls to achieve the purpose. You should only return the function call in tools call sections. If you decide to invoke any of the function(s), you MUST put it in the format of Put it in the format of [func1(params_name=params_value, params_name2=params_value2...), func2(params)] in <FUNCTION></FUNCTION> tags Here is a list of functions in JSON format that you can invoke. {FUNCTIONS HERE} The output format of all user requests are: <FUNCTION>[Put function callings (if any) here]</FUNCTION>
User	User request: {user request} Given Thinking Process or Analysis:{ROUND 1 THINKING}

Table 9: **Stage-2 Prompt** during our data construction process, which feeds generated reasoning following templates from Stage-1 back into the models to generate final function calling(s). Note that in this step, we do not feed the template into the prompt.

Stage-3: Training Sample Filtering Prompt	
Role	**Content**
System	You are an expert in composing functions. You are given a question, a set of possible functions and the ground truth function call(s), and a candidate for the correct function call. Based on the question and the ground truth function call(s), you will need to identify if the provided candidate is also a correct alternative function calling to the ground truth function callings to the question and provided functions. You must pay attention to the functions and parameters were used, if the candidate provide different numbers of functions from the ground truth, it is totally incorrect Here is a list of functions in JSON format that you can invoke. {FUNCTIONS HERE} You must only output one of the two label ['CAN REPLACE GROUND TRUTH','TOTALLY INCORRECT']. The output format of all user requests are: [PUT YOUR LABEL ONLY HERE]
User	User request: {user request} Ground Truth Function Calling:{GROUND TRUTH} Candidate Function Calling: {MODEL FUNCTION CALL FROM ROUND 2}

Table 10: Verification Prompt for Stage-3 during data construction process, which compares ground truth function calling and candidate function callings generated from Stage-2 for alternative solutions apart from ground-truth.

ToolGT Training Sample

Role	Content
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. Here is a list of functions in JSON format that you can invoke `{FUNCTIONS HERE}` Should you decide to return the function call(s). Put it in the format of [func1(params_name=params_value, params_name2=params_value2...), func2(params)] NO other text MUST be included.
User	I'm considering investing and I'd like to know what's happening in the market right now. Could you get me the top market trends in the US?

Assistant	**<THINKING>**
	1. The user is looking for information on current market trends, specifically the top market trends in the US. Based on the provided functions, the "Market Trends API" seems to be the most relevant as it is specifically designed to get the latest market trends.
	2. I will select the "Market Trends API" function to fulfill the user's request for market trends in the US.
	3. Now, I will examine the documentation for the "Market Trends API" function. The function requires the `trend_type` parameter, which specifies the type of trend to retrieve. The parameters `country` and `language` are optional but have default values.
	4. According to the documentation, the required parameters are:
	- `trend_type`: This is required and needs to be specified.
	- `country`: This is optional and defaults to "us".
	- `language`: This is also optional, with a default of "en".
	5. The user's request does not specify what type of market trend they want, but they mentioned wanting the "top market trends". This implies that they are likely interested in a specific category of trends. Since they did not specify, I will assume a common type of trend like "MARKET_INDEXES" as a default choice based on the context of "top market trends".
	6. The function call that can fulfill the user's request would look like this:
	`Market Trends API(trend_type="MARKET_INDEXES", country="us")`
	7. I will revalidate the composed function call to ensure it meets the requirements. The `trend_type` is specified, and the country defaults to "us" as intended.
	8. There are no errors identified in the function call. It satisfies the function documentation and fulfills the user's request for market trends in the US.
	</THINKING>
	<FUNCTION>
	`[Market Trends API(trend_type="MARKET_INDEXES", country="us")]`
	</FUNCTION>

Table 11: Training sample with reasoning following template used in our **ToolGT**