

GenTool: Enhancing Tool Generalization in Language Models through Zero-to-One and Weak-to-Strong Simulation

Jie He^{1*†}, Jennifer Neville^{2†}, Mengting Wan², Longqi Yang²,
Hui Liu², Xiaofeng Xu², Xia Song², Jeff Z. Pan¹, Pei Zhou^{2†}

¹ School of Informatics, University of Edinburgh, UK

² Microsoft Corporation

Abstract

Large Language Models (LLMs) can enhance their capabilities as AI assistants by integrating external tools, allowing them to access a wider range of information. While recent LLMs are typically fine-tuned with tool usage examples during supervised fine-tuning (SFT), questions remain about their ability to develop robust tool-usage skills and can effectively generalize to unseen queries and tools. In this work, we present **GenTool**, a novel training framework that prepares LLMs for diverse generalization challenges in tool utilization. Our approach addresses two fundamental dimensions critical for real-world applications: *Zero-to-One Generalization*, enabling the model to address queries initially lacking a suitable tool by adopting and utilizing one when it becomes available, and *Weak-to-Strong Generalization*, allowing models to leverage enhanced versions of existing tools to solve queries. To achieve this, we develop synthetic training data simulating these two dimensions of tool usage and introduce a two-stage fine-tuning approach: optimizing tool ranking, then refining tool selection. Through extensive experiments across four generalization scenarios, we demonstrate that our method significantly enhances the tool-usage capabilities of LLMs ranging from 1B to 8B parameters, achieving performance that surpasses GPT-4o. Furthermore, our analysis also provides valuable insights into the challenges LLMs encounter in tool generalization.

1 Introduction

Tool learning has emerged as a crucial capability for Large Language Models (LLMs), enabling them to expand their functionality through external tool integration (Huang et al., 2024b; Tang et al., 2023; Patil et al., 2023). By interfacing with external tools, LLMs can dynamically access real-time

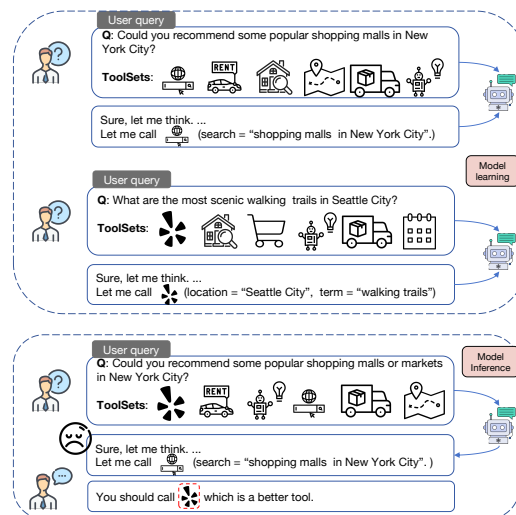


Figure 1: An example illustrating tool generalization challenges in selecting the most suitable tool for a user query. While the model was trained on tools like Yelp and Web-Search, and encountered the same query: “Could you ... in New York City?”, it struggles during testing to select the more appropriate Yelp tool over Web-Search for the same query during test.

information, validate responses against external knowledge bases, such as knowledge graphs (Pan et al., 2017b,a), and improve outputs through iterative feedback (Deng et al., 2023; Wang et al., 2024). This capability fundamentally enhances LLMs’ ability to process and respond to real-world information beyond their pre-trained knowledge (Pan et al., 2023).

The current paradigm for tool utilization follows a standard workflow (Qu et al., 2024): given a user query and available tool documentation, LLMs identify appropriate tool, extract required parameters, obtain information from the tool and synthesize final outputs. Two primary approaches have been developed to enable this capability: in-context learning, which leverages tool documentation and examples within the model’s context (Hsieh et al., 2023), and fine-tuning on collected tool usage data (Qin et al., 2024; Mekala et al., 2024). However,

*Work was done at Microsoft.

†Corresponding author: j.he@ed.ac.uk, jen-neville@microsoft.com, pei.zhou@microsoft.com.

both approaches face fundamental limitations. In-context learning is constrained by context length restrictions, preventing comprehensive tool understanding (Kim et al., 2023; Paranjape et al., 2023). Fine-tuning methods risk over-fitting to specific tools and usage patterns, compromising generalization to unseen tools and queries. These limitations pose significant challenges for real-world applications, where new tool categories and usage patterns constantly arise as shown in Figure 1.

To tackle these challenges, we introduce **GenTool**, a novel framework designed to enhance LLM’s tool generalization capabilities across two core dimensions: **zero-to-one generalization**, where a query initially has no available useful tool, and **weak-to-strong generalization**, where models must adapt from a weak tool which is ineffective in fully addressing the query to a strong one which can help provide an answer that perfectly matches the query. Through these dimensions, we identify four evaluation scenarios: *seen_query_unseen_tool*, *seen_query_seen_tool*, *unseen_query_unseen_tool*, and *unseen_query_seen_tool*.

To enable effective training across the two important dimensions, we developed a high-quality synthetic training dataset. This dataset comprises 834 new synthetic tools and their descriptions. For each tool, we generated 10 diverse queries, resulting in 8,515 distinct queries and 33,286 high-quality training samples. Each sample includes structured query-tool pairs and detailed parameter information. Additionally, we complement our data augmentation strategy with a novel two-stage fine-tuning strategy that explicitly teaches models to rank tools by capability before selection, enabling systematic understanding of tool relationships rather than simple query-tool mappings.

Our extensive evaluation demonstrates GenTool’s effectiveness across various model architectures including LLaMA 3.2 1B, LLaMA 3.1 8B, Mistral 7B, and Phi 3B. GenTool achieves state-of-the-art performance across all metrics, significantly outperforming both tuning-free and tuning-based baselines, including GPT-4o, with a 14.28% improvement in tool selection accuracy. Moreover, GenTool exhibits exceptional generalization capabilities across our four evaluation scenarios. Additionally, ablation studies validate the importance of our ranking mechanism, where its removal leads to performance degradation of up to 10.89%.

Our main contributions are summarized as follows:

1. We propose GenTool, a novel framework that enhances LLMs’ generalization capabilities on unseen tools through structured simulation of two critical dimensions. GenTool achieves state-of-the-art performance across all metrics, significantly outperforming various strong baselines.
2. We introduce a two-stage fine-tuning strategy combining tool selection with ranking supervision, enabling enabling robust tool generalization through better understanding of functional differences between similar tools.
3. We provide comprehensive empirical analysis revealing key insights into tool generalization, including the limitations of data scaling and the importance of diverse training settings, evidenced by a 67% performance drop when training exclusively on test-set-similar cases.

2 Problem Definition: Generalization in Tool Learning

Our framework aims to improve LLM generalization across diverse query-tool combinations. This section defines the tool generalization problem and presents a systematic framework for its evaluation.

2.1 Preliminaries

To analyze tool learning, we first define the core components of our framework. Each instance d in our dataset is represented as a tuple (T_s, q, y, t) , where the toolset T_s consists of k distinct tools, q the query submitted to the model, $t \in T_s$ the ground truth tool required to resolve the query, and y the tool-calling information specifying the tool name, parameter names, and their corresponding parameter values.

Each tool t_i within T_s is equipped by metadata that includes its name, descriptions of its parameters, names of expected results, and descriptions of those results. This formalization provides a foundation for assessing the model’s ability to generalize across varying combinations of queries and tools, establishing the groundwork for our framework.

2.2 Four Generalization Scenarios

To systematically evaluate a model’s generalization capabilities, we define four scenarios based on the familiarity of queries and tools during training.

Seen Query and Unseen Tool The training set contains a sample $d_{\text{train}}^1 = (T_s^1, q_1, t_1, y_1^1)$, while the test set includes a sample $d_{\text{test}}^1 =$

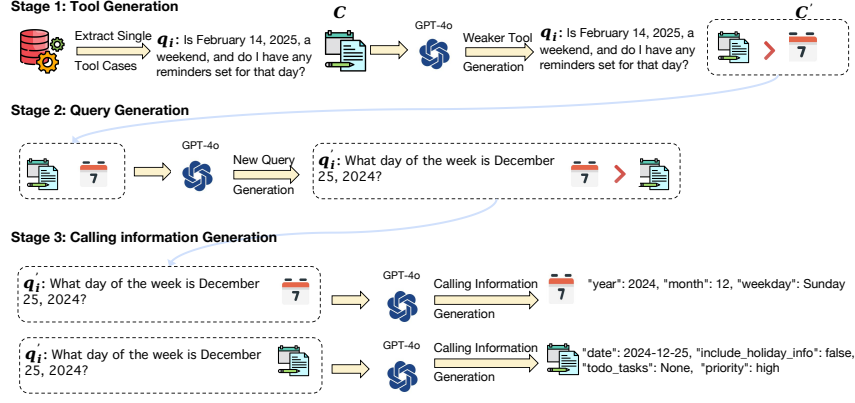


Figure 2: The construction process for synthetic data to simulate the generalization process involves three steps: First, existing datasets are utilized to create new tools. Next, diverse instructions guide the generation of matching queries for these new tools. Finally, corresponding invocation details are created for various tool-query combinations.

(T_s^2, q_1, t_2, y_1^2) . The query q_1 is identical across training and test sets, but the ground truth tool t_2 in the test case is not present in any training toolsets T_s , making q_1 a *seen query* and t_2 an *unseen tool*.

Seen Query and Seen Tool Both the query and the ground truth tool are present during training. The training set contains $d_{\text{train}}^1 = (T_s^1, q_1, t_2, y_1^2)$, and $d_{\text{train}}^2 = (T_s^1, q_2, t_1, y_1^1)$ and the test set includes $d_{\text{test}}^1 = (T_s^1, q_1, t_1, y_1^1)$. q_1 and t_1 are *seen*.

Unseen Query and Unseen Tool Neither the query nor the tool has been observed during training. The training set includes (T_s^1, q_1, t_1, y_1^1) , while the test set contains (T_s^2, q_2, t_2, y_2^2) , making q_2 and t_2 both *unseen*.

Unseen Query and Seen Tool The test query is unseen, but the ground truth tool appears in training. The training set includes (T_s^1, q_1, t_1, y_1^1) , and the test set has (T_s^1, q_2, t_1, y_2^1) . Here, q_2 is *unseen*, and t_1 is *seen*.

These scenarios provide a structured framework for assessing model generalization across queries and tools, offering valuable insights into robustness and adaptability of tool learning models in real-world applications.

3 GenTool : Synthetic Data Generation for Generalization Simulation

The gap in generalization remains challenging to address with current data (Qin et al., 2024; Tang et al., 2023), as models tend to learn biases from the tools available in the training set. To tackle this issue, we propose GenTool, a synthetic data generation framework that simulates generalization scenarios during the training phase.

Our approach is motivated by a key observation: when a model only has access to a basic calendar tool C' , it defaults to using it. However, when presented with both C' and a better tool, C , the model should recognize and select C for complex queries. We employ GPT-4o to generate additional tools that maintain the same format as the original tools in our dataset. For each original tool t , we generate weaker variants that have limited functionality while preserving the API structure.

Figure 2 illustrates our data generation pipeline, which consists of three primary components: tool generation, query generation, and calling information synthesis. Each component is designed to create diverse, high-quality training instances that specifically target generalization capabilities. Details for prompts and data quality checking, please refer to App. A.

3.1 Tool Generation

We build upon the UltraTool (Huang et al., 2024a) dataset, comprising 5,000 high-quality user queries refined by human annotators. For our initial investigation, we focus on the 800 single-tool usage cases (**original data**). This scope allows us to establish fundamental principles of tool generalization before tackling multi-tool interactions explored in works like MetaTool (Huang et al., 2024c).

For each query and its gold tool description, We use GPT-4o to generate two “weak tools” which is ineffective in fully addressing the query and fails to provide an answer that perfectly matches the query. For example, for the query “What are the best restaurants in New York City, and what cuisines do they serve?”, a weak tool may return the top 10 restaurants but not provide cuisines infor-

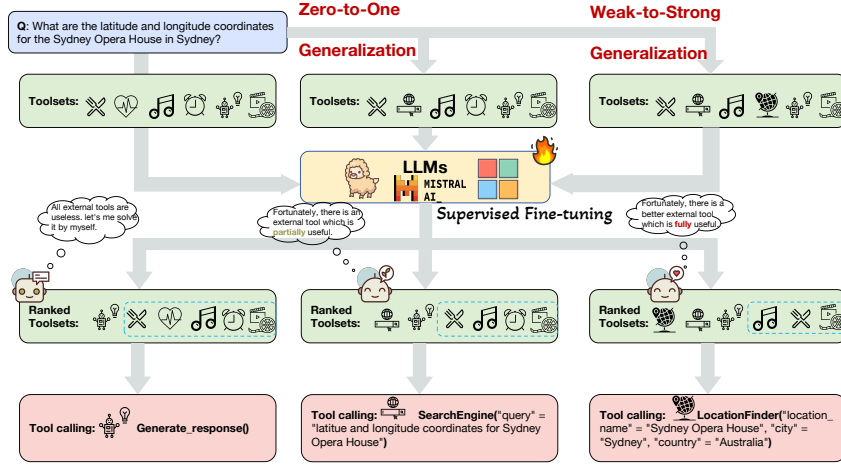


Figure 3: Overview of the GenTool Framework for Tool Learning and Generalization. Initially, the model handles a query by defaulting to `generate_response` when no suitable tool is available. Next, when a relevant tool, `web_search`, is added to the toolset, the model selects `web_search`, demonstrating zero-to-one generalization training. Later, upon adding `map_search`, the model demonstrates weak-to-strong generalization by correctly ranking and selecting it over `web_search` and other alternatives.

mation, whereas a strong tool could provide both the top 10 restaurants and their cuisines. In this case, the strong tool should be called. This process ensures consistent API formatting while varying tool capabilities.

3.2 Query Generation

To capture diverse generalization scenarios, we generate tailored queries for each weak tool. For each tool pair (gold tool t and weak tool t'), we prompt GPT-4o to generate 10 queries meeting two criteria: 1) solvable by the weak tool t' , and 2) only partially addressable by the gold tool t . This approach forces the model to distinguish tools based on capabilities, and the selection of 10 queries per tool pair provides sufficient variation while maintaining dataset manageability.

3.3 Calling Information Generation

Accurate tool invocation is crucial for training. We employ GPT-4o to generate calling information for each query-tool combination. For a given query q and a tool t , GPT-4o generates the necessary parameter names and corresponding parameter values required to invoke t , ensuring the dataset reflects the realistic process of tool invocation.

4 GenTool: Training and Evaluation Framework

This section presents our training and evaluation methodology as shown in Figure 3. We first introduce two generalization dimensions: *zero-to-one*

and *weak-to-strong* generalization. We then describe our test instances construction for evaluating the generalization scenarios outlined in Section 2.2. Finally, we propose a two-stage output mechanism to enhance tool selection accuracy.

4.1 Training Strategy for Two Generalization Dimensions

Our training addresses two real-world generalization dimensions: *zero-to-one* and *weak-to-strong*. We first describe our toolset generation process, which is fundamental to both frameworks.

4.1.1 Toolset Generation

For each query q , the model is provided with a toolset T_s consisting of k potential candidate tools, including the gold tool t required for the query. The construction process involves: 1) Computing embeddings for all tools in the dataset using the text-embedding-ada-003 model (OpenAI, 2023); 2) For each (q, t) pair, retrieving the $k - 1$ most similar tools based on cosine similarity for T_{gold} to generate the T_s ; 3) Ensuring diversity by excluding redundant tool-query pairs in which tools already paired with the same query during retrieval. Considering the input length limitation of the model, we set $k = 5$. To handle cases where no suitable tool exists in the toolset, we introduce `generate_response` as an additional tool, which enables the model to generate direct responses when necessary.

4.1.2 Zero-to-One Generalization Training

Given the dataset $QT = \{(q_1, t_1, q'_1, t'_1), \dots, (q_n, t_n, q'_n, t'_n)\}_{i=1}^n$, we first sample a subset $QT_{\text{pure_tr}} = \{(q_i, t_i, q'_i, t'_i)\}_{i=1}^k$ to construct training instances simulating *zero-to-one* and *weak-to-strong* transitions. The remaining portion of the dataset is defined as $QT_{\text{test}} = \{(q_j, t_j, q'_j, t'_j)\}_{j=k+1}^n$ serves as our evaluation set for testing generalization under four distinct scenarios (See 4.2).

For each $\{q, t, q', t'\}_i \in QT_{\text{pure_tr}}$, we constructed two training examples:

1. $d_{\text{train}}^1 = (T_s^N, q_1, \text{None}, y_N)$, where $t = \text{None}$, it indicates that the toolsets do not contain a tool matching the given query. In this case, $y_N = \text{generate_response}()$.
2. $d_{\text{train}}^2 = (T_s^1, q_1, t_1, y_1^1)$, where the model should select t_1 .

By including both d_{train}^1 and d_{train}^2 in the training set, the model learns to distinguish between cases where no tools in the toolset can solve the query and cases where a valid tool exists, thereby simulating *zero-to-one* generalization. Similarly, q' and t' follow similar construction patterns for *zero-to-one* generalization training example.

4.1.3 Weak-to-Strong Generalization Training

As described in Section 3, for each $\{q, t, q', t'\}_i \in QT_{\text{pure_tr}}$, we have a *weak tool* t' and a corresponding query q' . Using the synthesized data, we create:

1. $d_{\text{train}}^1 = (T_s^{1'}, q_1, t'_1, y_1^{1'})$, where the model needs to select t'_1 , as only this weak tool is available.
2. $d_{\text{train}}^2 = (T_s^{1+}, q_1, t_1, y_1^1)$, where $T_s^{1+} = \{t_1, t'_1, \dots\}$, the model needs to select the strong tool t_1 , with both weak and strong tools present.

By including d_{train}^1 and d_{train}^2 in the training set, the model learns to prioritize strong tools over weak ones, simulating *weak-to-strong* generalization.

4.2 Test Instances Construction

Building upon the four scenarios defined in Section 2.2, we now describe how to construct instances from QT_{test} for each scenario. Note that the `generate_response` tool remains available across all scenarios.

Seen_Query_Unseen_Tool For a given query-tool cluster $C_2 = \{q_2, q'_2, t_2, t'_2\} \in QT_{\text{test}}$, we first construct a training instance with no available

tools:

$$d_{\text{train}}^1 : (T_s^N, q_2, \text{None}, y_N)$$

where the model must use `generate_response`. For testing, we introduce unseen tools and construct:

$$\begin{aligned} d_{\text{test}}^1 &: (T_s^2, q_2, t_2, y_2^2), \\ d_{\text{test}}^2 &: (T_s^{2'}, q_2, t'_2, y_2^{2'}) \end{aligned}$$

Additionally, we use another cluster $C_3 = \{q_3, q'_3, t_3, t'_3\} \in QT_{\text{test}}$ to create a training instance with only weak tools:

$$d_{\text{train}}^1 : (T_s^{3'}, q_3, t'_3, y_3^{3'})$$

and test with both weak and strong tools:

$$d_{\text{test}}^1 : (T_s^{3+}, q_3, t_3, y_3^3)$$

In this case, the strong tool remains unseen in the training data,

Unseen_Query_Unseen_Tool Using cluster C_3 , we extend testing to unseen query and unseen tool:

$$d_{\text{test}}^1 : (T_s^3, q'_3, t_3, y_3^3)$$

where both q'_3 and t_3 are absent from training data.

Seen_Query_Seen_Tool Using cluster $C_4 = \{q_4, q'_4, t_4, t'_4\} \in QT_{\text{test}}$, we construct training instances:

$$\begin{aligned} d_{\text{train}}^1 &: (T_s^{4'}, q_4, t'_4, y_4^{4'}), \\ d_{\text{train}}^2 &: (T_s^4, q_4, t_4, y_4^4) \end{aligned}$$

and test with expanded toolsets:

$$d_{\text{test}}^1 : (T_s^{4+}, q_4, t_4, y_4^4)$$

Unseen_Query_Seen_Tool Using cluster C_3 , we construct test instances:

$$\begin{aligned} d_{\text{test}}^1 &: (T_s^{3'}, q_3, t'_3, y_3^{3'}) \\ d_{\text{test}}^2 &: (T_s^{3'}, q_3, \text{None}, y_N) \end{aligned}$$

where q_3 is unseen while the tools have been exposed during training.

4.3 Two-stage Output Training

Efficient tool usage in real-world applications requires prioritizing tools in the toolset. Existing fine-tuning approaches often directly output the selected tool without considering their relative priority. For instance, even when the gold tool is included in the

toolsets, `generate_response` still take precedence over other irrelevant tools. To address this, we propose a two-stage process: (1) Generating a ranked list of tools in the toolset. (2) Providing detailed calling information for the highest-ranked tool.

Our ranking system considers three distinct cases:

1. **No useful tools:** `generate_response` ranks first, followed by other tools.
2. **Single useful tool:** useful tool > `generate_response` > others.
3. **Multiple tools:** strong tool > weak tool > `generate_response` > other tools.

The detailed prompt design for this task is available in Appendix C.

5 Experimental Setup and Dataset

5.1 Evaluation Metrics

We evaluate model performance through a comprehensive assessment of tool invocation accuracy, comparing predicted outputs against ground truth across four critical dimensions: tool selection accuracy, parameter name identification, parameter value matching, and syntactic format correctness. For detailed explanations, please refer to App. D.

5.2 Baseline

We evaluate our approach using four models: **Llama-3.1-8B-Instruct**, **Llama-3.2-1B-Instruct** (Dubey et al., 2024), **Mistral-Instruct-7B-v0.3** (Jiang et al., 2023), and **Phi-3.5-3B** (Abdin et al., 2024). For simplicity, we omitted the name “instruct” in some experimental results. We compare two types of methods: **tuning-free** approaches using GPT-3.5, GPT-4o-mini, GPT-4o (OpenAI et al., 2024), ToolLLama (Qin et al., 2024), and GPT4Tools (Yang et al., 2023); and **tuning-based** methods including **Original: fine-tuning on seed data** (cf. Section 3.1) and **Half-Sample: using single examples from paired data** (e.g., selecting either the “zero” or “one” example from zero-to-one pairs). For model implementation details and data statistics, please see App. C.

6 Experimental Results

6.1 Overall Performance

Comprehensive evaluation results (Table 1) show GenTool’s superior performance across all metrics. Models fine-tuned with GenTool consistently outperform GPT-based baselines, with particularly

Method	Tool Selection	Parameter		Format Accuracy
		Name	Value	
Baselines				
GPT-3.5	48.61	70.34	59.25	99.30
GPT-4o Mini	58.80	78.86	69.05	99.93
GPT-4o	75.87	72.00	64.09	99.81
GPT4Tools	22.34	28.19	22.47	99.53
ToolLLaMA	17.73	24.59	18.43	99.38
Llama-3.2-1B-Instruct				
Zero-shot	29.02	0.08	0.07	40.12
Original	31.63	35.67	31.12	99.88
Half-Sample	61.52	58.48	52.19	99.92
GenTool	86.31	76.92	68.81	99.88
Phi-3.5-3B-Instruct				
Zero-shot	24.15	30.06	19.14	68.19
Original	55.49	58.58	52.63	99.90
Half-Sample	80.77	75.23	68.22	99.85
GenTool	87.37	83.48	75.62	99.83
Mistral-Instruct-7B-v0.3				
Zero-shot	45.33	66.27	56.07	97.89
Original	18.28	19.71	17.18	99.25
Half-Sample	51.00	49.74	44.52	99.78
GenTool	80.28	80.30	72.43	99.68
LLaMA-3.1-8B-Instruct				
Zero-shot	40.52	60.71	50.27	99.96
Original	48.09	42.67	38.03	99.82
Half-Sample	60.47	73.39	66.53	99.72
GenTool	90.15	86.05	77.64	99.45

Table 1: Overall tool-calling results. All baseline models are evaluated in a zero-shot setting. For different LLMs, the **original** and **half-sample** settings represent results obtained through fine-tuning.

notable improvements in *Tool Name Selection*. The LLaMA3.1-8B model fine-tuned with GenTool outperforms GPT-4o by 14.28%, demonstrating GenTool’s strong potential in selecting the appropriate tool. This substantial gain suggests that our structured training approach better captures the underlying patterns of tool selection.

For *tuning-free* methods, distinct performance patterns across model scales. Smaller models (LLama-3.1-1B, Phi-3.5-3B) struggle with instruction following, while models like GPT4Tools and ToolLLaMA achieve high format accuracy but lower performance than zero-shot baselines, probably due to domain shifts between their training datasets and UltraTool specific requirements, highlighting the challenges in cross-domain tool generalization.

For *tuning-based* approaches, *Half-Sample* outperforms *Original*, highlighting the benefits of our training data. In addition, performance shows no consistent correlation with model size, indicating it might not be critical for tool generalization.

6.2 Performance across Different Generalization Scenarios

We examine GenTool’s performance across different generalization scenarios, excluding *tuning-free baselines* as they treat all inputs as unseen. Figure 4 shows that GenTool excels particularly with seen queries. For the *Tool Selection* task, our GenTool demonstrates substantial improvements over the strongest baseline, achieving average gains of

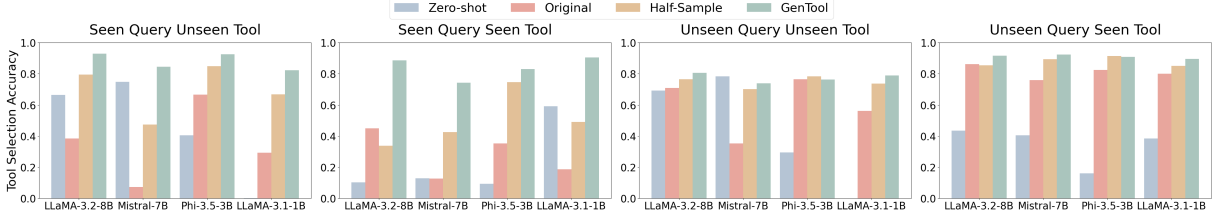


Figure 4: Detailed results for different test set scenarios. GenTool consistently outperforms all baselines.

Model	Tool Selection	Parameter	
		Name	Value
LLaMA-3.1-8B (Ours)	90.15	86.05	77.64
– w/o Rank	83.68	84.58	76.32
Mistral-7B (Ours)	80.28	80.30	72.43
– w/o Rank	69.39	71.49	63.98

Table 2: Ablation Study Results: Impact of Removing the Tool Ranking Task During Fine-Tuning. We instruct the model to directly output the most suitable tool-calling information.

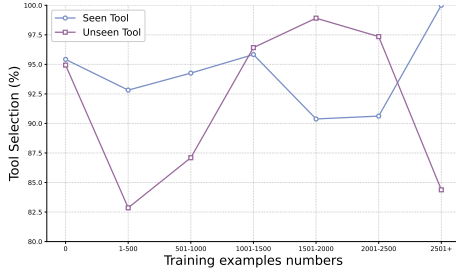


Figure 5: The relationship between the number of training examples related to the test set’s gold tool and the test set’s accuracy. The blue line represents seen tools, while the purple line denotes unseen tools.

18.4% in the *Seen_Query_Unseen_Tool* and 34.1% *Seen_Query_Seen_Tool* scenarios.

The framework maintains strong performance even in challenging unseen query scenarios. In the *Unseen_Query_Unseen_Tool* and *Unseen_Query_Seen_Tool* scenarios, GenTool consistently outperforms all baselines. Our approach outperforms the best baseline (half-sample) by 2.8% and 3.2% respectively. This performance suggests successful mitigation of overfitting to seen queries, a common challenge in tool learning systems.

6.3 Ablation Study

To validate the effectiveness of our ranking mechanism, we conduct systematic ablation experiments examining its contribution to overall performance. Table 2 reveals that removing the ranking component leads to consistent performance degradation across all metrics. The impact is particularly obvious in Tool Selection accuracy, where LLaMA-3.1-8B and Mistral-7B models show significant decreases of 6.47% and 10.89%, respectively.

7 Empirical Analysis

In this section, we conduct comprehensive experiments to understand the factors influencing tool generalization and the limitations of models. Our analysis focuses on two key aspects: examining how different training data compositions affect model performance and investigating the fundamental characteristics of model capabilities.

7.1 Impact of Related Examples Quantity

To investigate how related training examples influence model performance, we analyze the correlation between the number of related examples and test performance. We hypothesize that performance improves with increased exposure to related tools in the training set. Using the text-embedding-3-large model (OpenAI, 2023), we calculate semantic similarities for embeddings between gold tools in the training and test sets. For a test tool i , if a training tool j has a cosine similarity greater than 0.5, j is considered related to i .

Figure 5 reveals distinct trends for Seen and Unseen Tools. For Unseen Tools, we observe that the accuracy rises from 82% to 98%, then drops to around 84%. This non-monotonic pattern likely reflects a trade-off between memorization and generalization. Initially, a small number of superficially related tools may confuse the model, leading to suboptimal performance. As more relevant tools are introduced, the model gains richer exposure and generalizes better, reaching peak performance. However, when the number of related examples exceeds a certain threshold (e.g., 2000), accuracy begins to drop again. We hypothesize that this late-stage decline is due to overfitting and semantic redundancy. Specifically, with too many related examples—especially if they only share superficial patterns with the unseen tool but lack its full functionality—the model may overfit to a narrow and potentially misleading distribution, thereby reducing its robustness to truly novel tools. Moreover, after a certain point, additional related examples may cease to provide new information and instead

Model	Tool Selection	Parameter	
		Name	Value
GenTool	90.15	86.05	77.64
Weak-to-Strong	63.36	82.72	73.65
Zero-to-One	71.01	62.37	55.87

Table 3: Results of the LLaMA-3.1-8B-Instruct model trained in a single generalization simulation scenario. **Weak-to-strong** indicates the model is trained only on the weak-to-strong-related dataset, while **zero-to-one** indicates the model is trained only on the zero-to-one-related dataset.

reinforce spurious correlations or introduce noise, further impairing generalization.

In contrast, for Seen Tools, the tool selection accuracy remains consistently high (above 90%) across varying numbers of related examples. This suggests that once a tool is directly observed during training, additional related data continues to benefit performance, with only minor fluctuations.

7.2 Contribution of Different Pair Types

We evaluate the individual impact of *Zero-to-One* and *Weak-to-Strong* pairs on model generalization by fine-tuning models exclusively on either Zero-to-One or Weak-to-Strong pairs. Table 3 shows that *weak-to-strong* pairs enhance parameter name accuracy but lead to lower tool selection accuracy, excelling when tools are available but struggling to identify when no tool is applicable. In contrast, fine-tuned models with *zero-to-one* pairs exhibit higher tool selection accuracy, indicating better capability in identifying scenarios where tool usage is inappropriate. This suggests both pair types are essential for comprehensive tool generalization.

7.3 Training with Test-Related Examples Only

To examine whether models understand tools during learning, we train models exclusively on test-relevant tools and queries. Table 4 reveals significant performance deterioration across all models, especially fine-tuned models. For instance, Mistral-7B’s accuracy in *Tool Selection* drops from 45.33% to 11.56%. These results indicate that models may not fully understand tools or queries, and slight variations in tool usage scenarios lead to sharp accuracy declines. This aligns with observations in RoTuning (Ye et al., 2024).

7.4 Top Ranking Performance

Our ranking-based mechanism requires models to rank tools and generate detailed calls for the top-ranked option. As shown in Table 5, fine-tuned

Model	Tool Selection	Parameter	
		Name	Value
Llama-3.2	16.32 (-69.99)	18.55 (-58.37)	16.43 (-52.38)
Phi-3.5-3B	27.99 (-59.38)	35.91 (-47.57)	32.07 (-43.55)
Mistral-7B	11.56 (-68.72)	16.36 (-63.94)	14.32 (-58.11)
Llama-3.1-8B	17.11 (-73.04)	24.57 (-61.48)	21.51 (-56.13)

Table 4: Performance comparison of models trained exclusively on test-relevant examples. **green** and **blue** values represent the differences compared to the GenTool method respectively.

Model	(1)	(2)
Llama-3.2-1B-Instruct	99.81	78.45
Phi-3.5-3B	99.82	78.56
Mistral-Instruct-7B-v0.3	99.53	71.48
Llama-3.1-8B-Instruct	99.41	80.09

Table 5: Tool ranking analysis: (1) consistency between top-ranked in the first task and re-predicted tools in the second task; (2) accuracy of useful relevant tool ranking relative to `generate_response`.

models achieve nearly 100% accuracy in executing the top-ranked tool, but struggle to distinguish useful tools from irrelevant ones, with consistency below 80%. This highlights a challenge in filtering irrelevant tools despite strong *Tool Selection* accuracy.

8 Related Work

8.1 Tool Learning with LLMs

Recent studies have enhanced large language models (LLMs) with tools such as code-related APIs (Patil et al., 2023), retrievers (Shen et al., 2025), mathematical functions (Gou et al., 2024), and feedback mechanisms to refine outputs and reduce hallucinations (Dhuliawala et al., 2024). For example, Mekala et al. (2024) emphasizes tool-based verification for reliable answers, while Xu et al. (2024) enhance tool retrieval capabilities with iterative feedback from LLMs. Gao et al. (2024) propose progressive learning to improve tool usage across tasks. In parallel, ToolGen (Wang et al., 2025) introduces a unified generation-based framework that embeds tool knowledge directly into the LLM’s vocabulary using virtual tokens, enabling it to retrieve and invoke tools without an external retriever. However, these approaches lack a comprehensive evaluation of tool generalization. Our work addresses this by defining the tool generalization problem and incorporating diverse scenarios into training.

8.2 Synthetic Data Generation

LLMs’ generative capabilities have facilitated synthetic data generation, reducing annotation costs

while maintaining consistency (Zelikman et al., 2022; Honovich et al., 2023; He et al., 2025). For tool learning, Huang et al. (2024a) and Huang et al. (2024c) generate complex queries to evaluate tool usage, while Qin et al. (2024) and Tang et al. (2023) use APIs and instruction-response pairs to improve fine-tuning. However, these methods often overlook scenario complexity, limiting generalization performance. Our approach extends these methods by synthesizing data tailored to enhance performance across diverse tool generalization scenarios.

9 Conclusion

This work addresses the generalization challenges of LLMs in tool learning by proposing GenTool, a novel approach simulating the generalization process from two dimensions: *Zero-to-One* and *Weak-to-Strong Generalization*. GenTool further enhances LLMs’ tool selection by guiding them to output ranked tool lists.

Extensive experiments demonstrate that our method outperforms GPT-4o, achieving superior results across all generalization scenarios when compared to strong tuning-based and tuning-free baselines. Further analysis not only highlights that GenTool consistently delivers better performance but also reveals a key insight: existing LLMs rely heavily on memorizing tools rather than understanding their use. This indicates a significant gap between current LLMs and the goal of developing truly intelligent agents capable of effective tool utilization. We hope this research inspires further advancements in LLMs’ tool-learning capabilities.

Limitation

Model Scale Our experimental scope was constrained to backbone models under 8 billion parameters due to computational limitations. While GenTool has demonstrated promising results with these smaller models, its effectiveness with larger-scale architectures remains unexplored. This limitation is particularly noteworthy given that our training data was synthesized using GPT-4o. An important avenue for future research would be investigating whether GenTool’s approach can enhance the fine-tuning capabilities of larger models, including GPT-4o itself.

Focusing on Single-query and Single-tool Scenarios Our work currently addresses single-query, single-tool scenarios, which represent 44.26% of real-world applications according to

the ToolBench dataset (Qin et al., 2024). While this focus enabled us to conduct **the first** comprehensive investigation of tool learning generalization across two dimensions examining four distinct scenarios: *seen_query_unseen_tool*, *seen_query_seen_tool*, *unseen_query_unseen_tool*, and *unseen_query_seen_tool*, we acknowledge that real-world applications often require more complex multi-step planning (Vyas et al., 2025). Our work establishes a foundation for understanding generalization in simpler contexts, paving the way for future research into more complex orchestration challenges involving multiple queries and tools.

Bias of Synthetic Data While using a single LLM for dataset construction is common in related research (Huang et al., 2024c; Gao et al., 2024), our choice of GPT-4o for data synthesis may introduce inherent biases. However, although utilizing multiple LLMs could potentially reduce these biases, it would likely introduce variance in generation quality, potentially compromising evaluation stability. Given that our task primarily involves generating straightforward tools and queries, and that GPT-4o’s biases are not directly related to the tool learning generalization problem, we determined that using a single, high-quality model for data synthesis was the most appropriate approach for this work.

References

- Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, and . al Misha Bilenko et. 2024. [Phi-3 technical report: A highly capable language model locally on your phone](#). *Preprint*, arXiv:2404.14219.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. [Mind2web: Towards a generalist agent for the web](#). *Preprint*, arXiv:2306.06070.
- Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. [Chain-of-verification reduces hallucination in large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3563–3578, Bangkok, Thailand. Association for Computational Linguistics.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang,

- Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, and et al. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- 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](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18030–18038.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujia Yang, Nan Duan, and Weizhu Chen. 2024. [CRITIC: Large language models can self-correct with tool-interactive critiquing](#). In *The Twelfth International Conference on Learning Representations*.
- Jie He, Yijun Yang, Wanqiu Long, Deyi Xiong, Victor Gutierrez Basulto, and Jeff Z. Pan. 2025. [Evaluating and improving graph to text generation with large language models](#). 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 10219–10244, Albuquerque, New Mexico. Association for Computational Linguistics.
- Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2023. [Unnatural instructions: Tuning language models with \(almost\) no human labor](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14409–14428, Toronto, Canada. Association for Computational Linguistics.
- 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](#). *Preprint*, arXiv:2308.00675.
- Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024a. [Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4363–4400, Bangkok, Thailand. Association for Computational Linguistics.
- Tenghao Huang, Dongwon Jung, Vaibhav Kumar, Mohammad Kachuee, Xiang Li, Puyang Xu, and Muhao Chen. 2024b. [Planning and editing what you retrieve for enhanced tool learning](#). In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 975–988, Mexico City, Mexico. Association for Computational Linguistics.
- Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024c. [Meta-tool benchmark for large language models: Deciding whether to use tools and which to use](#). In *The Twelfth International Conference on Learning Representations*.
- Shadi Iskander, Sofia Tolmach, Ori Shapira, Nachshon Cohen, and Zohar Karnin. 2024. [Quality matters: Evaluating synthetic data for tool-using LLMs](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4958–4976, Miami, Florida, USA. Association for Computational Linguistics.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. 2023. [Mistral 7b](#). *Preprint*, arXiv:2310.06825.
- Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. [Language models can solve computer tasks](#). *CoRR*, abs/2303.17491.
- Ilya Loshchilov and Frank Hutter. 2019. [Decoupled weight decay regularization](#). In *International Conference on Learning Representations*.
- Dheeraj Mekala, Jason E Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. 2024. [TOOLVERIFIER: Generalization to new tools via self-verification](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5026–5041, Miami, Florida, USA. Association for Computational Linguistics.
- OpenAI. 2023. New and improved embedding model. <https://openai.com/blog/new-and-improved-embedding-model>. Accessed: 2024-11-25.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, and Suchir Balaji etc. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeliyanenko, Wen Zhang, Matteo Lissandrini, Russa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and Damien Graux. 2023. Large Language Models and Knowledge Graphs: Opportunities and Challenges. pages 1–38.
- J.Z. Pan, D. Calvanese, T. Eiter, I. Horrocks, M. Kifer, F. Lin, and Y. Zhao, editors. 2017a. *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Querying Answering*. Springer.
- J.Z. Pan, G. Vetere, J.M. Gomez-Perez, and H. Wu, editors. 2017b. *Exploiting Linked Data and Knowledge Graphs for Large Organisations*. Springer.
- 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](#). *Preprint*, arXiv:2303.09014.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.
- 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. 2024. [Tool learning with large language models: A survey](#). *Preprint*, arXiv:2405.17935.
- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. [Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters](#). In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20*, page 3505–3506, New York, NY, USA. Association for Computing Machinery.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, Zhongyang Li, YE QI, Yang Ren, Dandan Tu, and Jeff Z. Pan. 2025. GeAR: Graph-enhanced Agent for Retrieval-augmented Generation. In *In the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*.
- Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. [Toolalpaca: Generalized tool learning for language models with 3000 simulated cases](#). *Preprint*, arXiv:2306.05301.
- Kaustubh Vyas, Damien Graux, Yijun Yang, Sebastien Montella, Chenxin Diao, Wendi Zhou, Pavlos Vougiouklis, Ruofei Lai, Yang Ren, Keshuang Li, and Jeff Z. Pan. 2025. From an LLM Swarm to a PDDL-empowered Hive: Planning Self-executed Instructions in a Multi-modal Jungle. In *In the Proceedings of the International Conference on Learning Representations (ICLR 2025)*.
- Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. [LLMs in the imaginary: Tool learning through simulated trial and error](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604, Bangkok, Thailand. Association for Computational Linguistics.
- Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025. [Toolgen: Unified tool retrieval and calling via generation](#). *Preprint*, arXiv:2410.03439.
- Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. [Enhancing tool retrieval with iterative feedback from large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9609–9619, Miami, Florida, USA. Association for Computational Linguistics.
- Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. [GPT4tools: Teaching large language model to use tools via self-instruction](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. [RoTBench: A multi-level benchmark for evaluating the robustness of large language models in tool learning](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 313–333, Miami, Florida, USA. Association for Computational Linguistics.
- Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. [STar: Bootstrapping reasoning with reasoning](#). In *Advances in Neural Information Processing Systems*.

A Dataset generation and checking

A.1 Dataset Generation Prompts

Table 10, Table 11 and Table 12 present the prompts used to generate our dataset with GPT4o. Specifically:

- Table 10 contains prompts designed to generate *weak tools* based on existing query-tool pairs.
- Table 11 includes prompts for diversifying query generation based on *strong tools* and *newly generated weak tools*.
- Table 12 provides prompts for generating output information based on query and tool combinations.

Figure 6 shows the example that we used in Section 4.2 to explain how we generate the four generalization scenarios.

A.2 Human Verification

To ensure data quality, we implement a systematic verification process. We randomly sample 200 instances, including queries, strong tools, the generated weak tools, and the queries derived from weak tools. An expert annotator, with extensive experience in API design and natural language processing, evaluates a random sample of 200 instances including queries, strong tools, the generated weak tools, and the queries derived from weak tools. Inspired by Iskander et al. (2024), the verification process for each instance follow the criteria in Table 8.

Results in Table 8 show that over 90% of all fields are valid, significantly higher than the 33% parameter alignment error rate observed in datasets like ToolBench and ToolAlpaca (Iskander et al., 2024). These high verification scores demonstrate the effectiveness of our generation pipeline in producing high-quality, reliable training data for tool generalization tasks.

B Dataset Comparison

To further emphasize the distinctions between our work and existing approaches, Table 6 presents a comparison with recent studies that leverage LLMs for synthetic data generation followed by fine-tuning. Notably, our method employs the state-of-the-art GPT-4o model for data generation, ensuring high-quality data. Moreover, we tackle diverse

generalization scenarios, including complex compositional relationships between queries and tools, which remain unexplored in prior research. To further elevate the challenge of tool selection, we adopt a retrieval-based strategy to construct candidate toolsets, significantly testing the model’s ability to identify the correct tool. Our GenTool method achieves an accuracy of up to 90.15% on the tool selection task, demonstrating the effectiveness and superiority of our approach.

C Experiment Setup

C.1 Data Statistics

To validate the effectiveness of GenTool, we synthesized a dataset covering 22 domains, 1,633 tools, and 33,286 samples. We used 67% of the tool clusters to construct training sets for *zero-to-one* and *weak-to-strong* generalization and 33% of the clusters to simulate test scenarios. Detailed statistics are shown in Table 7.

C.2 Implementation Details

For our fine-tuning experiments, all model fine-tuning is done using the AdamW optimizer (Loshchilov and Hutter, 2019) with $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and no weight decay. The learning rate is set to $3e-5$, and we use a linear scheduler with a warm-up ratio of 0.1. The batch size is set to 4, with a training epoch of 4, and training examples are truncated to 3076 tokens. We optimize the model training using the DeepSpeed zero strategy (Rasley et al., 2020). The model training can be completed in 15 hours using four Nvidia A100 PCIe 80GB GPUs. Notably, we exclude the result information from calling T_{gold} , as the primary focus of this study is on the tool generalization problem rather than the execution of actual API calls.

In our fine-tuning process, we involve two tasks: *Tool Ranking* and *Tool Invocation*. The instructions are set as follows:

- The input to the model follows the format:
[Input Query: ... | Toolset: ...].
- The model’s gold output format is:

The output of the first task is: [Tool1, Tool2, Tool3]. The output of the second task is:

Method	Base Model	Dataset Construction	Generalization Scenario in Evaluation	Candidates Construction
Toolformer (Schick et al., 2023)	GPT-J-6B	In-Context Learning	1	Manually
GPT4Tools (Yang et al., 2023)	Vicuna-13B	Manually	1	Manually
ToolAlpaca (Tang et al., 2023)	Vicuna-13B	Simulation	1	Manually
ToolBench (Qin et al., 2024)	LLaMA-30B	Self-Instruct	1	Manually
ToolVerify (Mekala et al., 2024)	LLaMA-70B	Self-Instruct	2	Random
Confucius (Gao et al., 2024)	LLaMA-7B	Iterative Self-Instruct	2	Retrieval
STE (Wang et al., 2024)	GPT-3.5	Simulation	1	Manually
Ours	GPT-4o	Simulation	4	Retrieval

Table 6: **Dataset construction** refers to the method used to obtain the training dataset. **Generalization scenario in evaluation** indicates the number of generalization types covered during testing. **Candidates construction** specifies whether the candidate set is meticulously curated manually or generated using the same retrieval method as employed in real-world application scenarios.

Dataset	Tools amount	Instance amount	Avg. word input/output
Training	868	26,042	597/27
Test cases (1)	471	2,996	610/33
Test cases (2)	278	3,072	573/22
Test cases (3)	252	448	574/32
Test cases (4)	125	728	554/25

Table 7: The data statistics for training and testing sets constructed from synthetic data, where the test set is divided into the following four categories based on generalization scenarios: (1) Seen_query_unseen_tool (2) Seen_query_seen_tool (3) Unseen_query_unseen_tool (4) Unseen_query_seen_tool

Quality Review Question	Correctness Rate
Is the instruction a valid and well-formed query?	94%
Can the reference tool provide at least a partial solution to the query?	97%
Does the tool set selection comply with output specifications? (e.g., for “none” outputs, confirming no tool in the set can partially address the query)	97%
Are all tool call parameters correctly extracted or reasonably inferred from the instruction without missing or imagined values?	96%
All fields are valid	93%

Table 8: Quality Review Results for synthetic data

Table 9 shows the detailed prompt when we fine-tune the LLMs under the GenTool training framework.

D Detailed explanation of evaluation metrics

D.1 Tool Selection

We assess the model’s ability to discriminate and select the best tool by comparing the predicted tool name against the ground truth. This metric

directly evaluates the model’s capability to understand query contexts and match them with appropriate tools.

D.2 Parameter Name Identification

After selecting the appropriate tool, the model identifies the parameters required for the tool invocation based on the query. The output consists of the parameter names, and we compare the model’s output parameter names with the standard parameter names, focusing on both completeness and correctness.

D.3 Parameter Value Matching

After selecting the tool and the required parameters, the model parses the query to fill in the content needed for the parameters. The parameter value evaluation focuses on the model’s ability to extract and generate appropriate values from queries. Following Huang et al. (2024a), we employ a normalized Levenshtein distance-based scoring function. Given the model’s key-value format response (p_k, p_v) and the reference answer (y_k, y_v) , where keys p_k and y_k represent the steps and values p_v and y_v represent task-specific results, the score is defined as follows::

$$S = \begin{cases} F(p_v, y_v) & \text{if } p_k = y_k \\ 0 & \text{if } p_k \neq y_k \end{cases}$$

where S is the calculated score, and F represents the calculation function using normalized Levenshtein distance.

D.4 Format Correctness

Format correctness evaluates whether the model’s tool invocation output is free from syntax or matching errors and can be correctly parsed into the stan-

Tool Ranking and Invocation Process
You are a professional tool selection assistant. You will be given a query and a corresponding toolset. You have two consecutive tasks. First Task Rank the available tools in the toolset based on relevance. Output format is a list where each item is the name of the tool (from 'name' field in toolset list). Irrelevant tools should be placed after <code>generate_response()</code>, relevant tools before <code>generate_response()</code>. The order of irrelevant tools doesn't matter. Example format: [tool_name_1, generate_response(), tool_name_2, tool_name_3, tool_name_4, tool_name_5] Second Task Invoke the tool ranked first from the previous task and fill required parameter values. Parameter format: "parameter_name"="parameter_value", with multiple parameters separated by commas. Parameter names from 'properties' field under 'arguments', parameter values from user's query. If <code>generate_response()</code> ranks first, output <code>generate_response()</code>. Example format: ["tool_name_1"("parameter_name1"="parameter_value1", "parameter_name2"="parameter_value2")] Final Output Format (JSON): <pre>{ "The output of the first task": [], "The output of the second task": [] }</pre> Input: Query: {input_query} Toolset: {tools}

Table 9: Prompt for the GenTool methodology to rank and invoke the most relevant and effective tools from a given toolset

Tool Generation Process
You are a professional tool creation assistant. Given a #user query#, a set of #existing tools#, your task is as follows: 1. Create a #new tool# to solve the #user query#. 2. The #new tool# you create should be less effective than the #existing tools# in answering the #existing problem#. 3. Effectiveness can be judged from the following perspectives: 3.1 The match between the tool names and the problem. 3.2 The length of the tool descriptions and their match with the problem. 3.3 The names, descriptions, and number of tool parameters and their match with the problem. 3.4 The tool's returned results and descriptions and their match with the problem. 4. The format of the #new tool# should be the same as the tools in the #existing tools#. Below are examples of inputs for #user query#, and #existing tools#, and outputs for the #new tool#: Please generate #new tool# in JSON format based on the following #user query# and #existing tools#. Ensure the JSON string format is correct and do not output anything else. #user query#: {user_query} #existing tools#: {ex_tools} #new tool#

Table 10: Prompt for using GPT-4o to generate new weak tool

Query Generation Process
You are a professional question generation assistant. Based on the <code>#weak tool sets#</code> and <code>#strong tool sets#</code>, your task is to generate 10 questions that are suitable for answering using the <code>#strong tool sets#</code>. The requirements are as follows: The questions should be as detailed as possible. The questions should ideally only require the <code>#strong tool sets#</code> for answering, without needing assistance from other tools. Ensure that the examples provided are distinct from each other. You can use various sentence structures, such as commands or requests, and adjust the level of detail as needed. The questions can also be answered by the <code>#weak tool sets#</code>, but the <code>#weak tool sets#</code> should not match the questions as well as the <code>#strong tool sets#</code>. Based on the following <code>#weak tool sets#</code> and <code>#strong tool sets#</code>, generate 10 different questions in list format, following the example format. Do not output anything else. #weak tool sets#: {weak} #strong tool sets#: {strong} #output#:

Table 11: Prompt designed for generating diverse new queries.

Calling Annotation Prompt:
You are a professional tool matching assistant. Based on the <code>#user query#</code> and <code>#toolsets#</code>, your task is to perform tool matching for the <code>#user query#</code> and output a <code>#detailed execution plan#</code> with the following requirements: In the <code>#detailed execution plan#</code>, provide the tool names and their specific parameters. The form of parameter passing should be: "parameter name: parameter value". Ensure that the parameter names used come from the defined parameter name set in the corresponding tools, and ensure that the parameter values have sources, which can come from two parts: user instructions or some user-related information (such as personal information like name, ID number, account, password, etc.). Below are some examples of input <code>#user query#</code> and <code>#toolsets#</code> and the output <code>#detailed execution plan#</code>: #user query#: 'Please create a new file named "2023 October Work Log.txt" with the initial entry "Tasks Completed Today"' #toolsets#: {demonstrations_example_tool_set} #output#: file_write(file_path='Desktop/2023 October Work Log.txt', content='Tasks Completed Today') Please generate a <code>#detailed execution plan#</code> in JSON format based on the following <code>#user query#</code> and <code>#toolsets#</code>, following the format of the examples. Ensure the JSON string format is correct and do not output anything else. #user query#: {query} #toolsets#: {tools} #output#:

Table 12: Prompt for generating a detailed calling information based on user query and toolsets.

dard JSON format. This metric ensures the generated tool calls are executable.

E Additional analysis

E.1 Data Scaling Effects

We analyze model performance across different data percentages (10% to 100%). Figure 7 shows that performance improves significantly between 10-20% and 40-50% of data. Moreover, performance generally improves with more data, with varying gains across scenarios. The *seen_query_seen_tool* scenario benefits most from additional data, while the growth in *unseen_query_unseen_tool* slows down as the training data increases. Additionally, models achieve 78.61% tool selection accuracy with just 30% of data, comparable to the Half-Sample baseline, suggesting GenTool’s effectiveness comes from struc-

tured pair construction rather than data quantity.

E.2 Fine-grained Analysis for Transfer Learning

We evaluate the LLaMA-3.1-8B-Instruct model fine-tuned solely on weak-to-Strong pairs and zero-to-One pairs, respectively, and analyze their performance on the test set across four scenarios. As shown in Table 13, the Weak-to-Strong model performs better on *seen_query_unseen_tool* and *unseen_query_unseen_tool* scenarios. In contrast, the Zero-to-One model achieves better results on *seen_query_seen_tool* and *unseen_query_seen_tool* scenarios.

E.3 In-Context Learning Capability

We evaluate GenTool’s in-context learning using two demonstration pairs per test example (*Zero-to-One* and *Weak-to-Strong*), focusing on GPT-

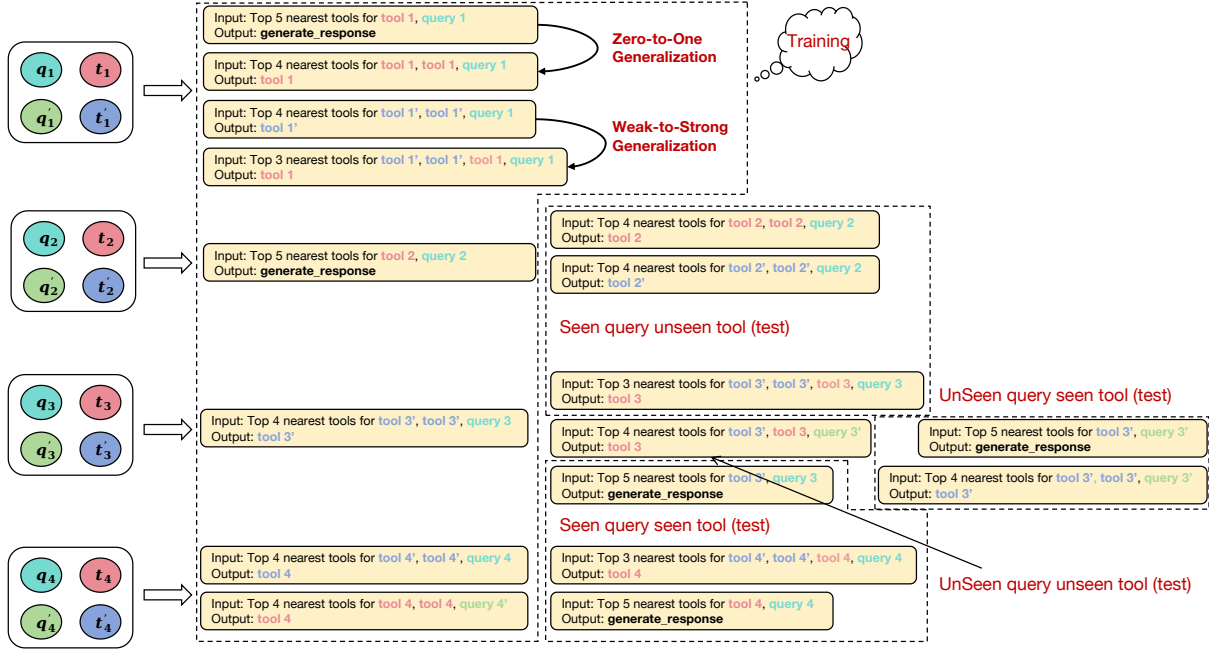


Figure 6: The detailed process of training and testing data division based on queries and the tool library: Different tool-query pairs play distinct roles. The first cluster focuses on constructing the training set. The second cluster is dedicated to building the training and testing sets for *seen_query_unseen_tool*. The third cluster provides training and testing sets for *seen_query_unseen_tool*, *unseen_query_unseen_tool*, and *unseen_query_seen_tool*. Finally, the fourth cluster constructs both the training and testing sets for *seen_query_seen_tool*.

Method	Tool Selection	Parameter		Format Accuracy
		Name	Value	
Seen Query, Unseen Tool				
GenTool	92.84	85.27	76.55	98.96
Weak-to-Strong	90.21	83.80	75.82	99.57
Zero-to-One	63.32	57.18	50.81	99.77
Seen Query, Seen Tool				
GenTool	88.57	90.30	80.89	99.71
Weak-to-Strong	31.62	42.79	41.72	98.73
Zero-to-One	74.47	65.91	56.96	99.71
Unseen Query, Unseen Tool				
GenTool	80.58	77.94	71.18	100.00
Weak-to-Strong	84.38	80.60	73.23	99.78
Zero-to-One	74.78	71.62	67.18	100.00
Unseen Query, Seen Tool				
GenTool	91.62	94.50	88.48	100.00
Weak-to-Strong	73.90	94.48	87.66	99.73
Zero-to-One	85.71	87.05	81.59	100.00

Table 13: Detailed results of the LLaMA-3.1-8B-Instruct model trained in a single generalization simulation scenario across different scenarios

series models due to input length constraints. Table 15 shows consistent improvements with GenTool. GPT-4 achieves the highest *Tool Name Selection* accuracy (84.27%), while GPT3.5 and GPT-

4o outperform the half-data baseline. Although GPT4o-mini lags in *Tool Name Selection*, it surpasses the baseline in *Param Name Matching* and *Param Value Matching* by 3.15% and 1.97%, re-

Method	Tool Selection	Parameter		Format Accuracy
		Name	Value	
gpt-3.5	31.75 (-16.86)	38.20 (-32.14)	31.99 (-27.26)	99.2 (-0.09)
gpt-4o mini	54.63 (-4.17)	62.53 (-16.33)	55.32 (-13.73)	100.0 (+0.07)
gpt-4o	68.96 (-6.91)	59.93 (-10.07)	53.31 (-10.78)	99.96 (+0.16)

Table 14: Results for GPT models with one demonstration that are directly relevant to the test examples. The green and blue values represent the differences compared to the model’s zero-shot performance, respectively.

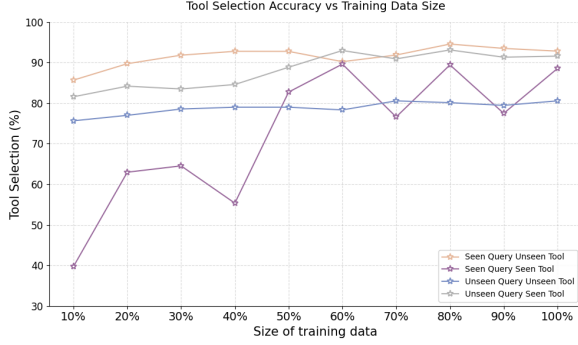


Figure 7: Effects of different proportions of training data on various test scenarios.

Method	Tool Selection	Parameter	
		Name	Value
gpt-3.5 (1)	48.61	70.34	59.25
gpt-3.5 (2)	49.65	52.55	44.61
k gpt-3.5 (3)	57.89	72.30	60.24
gpt-4o mini (1)	58.80	78.86	69.05
gpt-4o mini (2)	76.18	76.05	66.00
gpt-4o mini (3)	66.88	79.20	67.97
gpt-4o (1)	75.87	72.00	64.09
gpt-4o (2)	79.24	66.68	59.47
gpt-4o (3)	84.27	77.14	67.48

Table 15: Results for the GenTool method when applied to in-context learning. (1) represents results under the zero-shot setting, (2) represents results where half of the data from zero-to-one and weak-to-one pairs is randomly sampled for demonstrations, and (3) represents results where zero-to-one and weak-to-strong pairs are used for demonstrations.

spectively, showcasing GenTool’s effectiveness.

E.4 In-context Learning for Related Demonstrations

We further investigate the performance of GPT-series models when using training data constructed from test example-related pairs as demonstrations for in-context learning. As shown in Table 14, all models exhibit varying degrees of performance degradation across nearly all metrics. Notably, GPT-3.5 shows the most significant drop, consistent with findings in Section 7.4. This highlights that current large models, whether fine-tuned or used in in-context learning, fail to genuinely ac-

quire tool-use capabilities. Instead, they primarily rely on memorizing previously seen knowledge.

F Case Study

Table 16, 17 showcase two examples to demonstrate the effectiveness of GenTool .

- In the first example, the GenTool -finetuned LLaMA-8B model successfully outputs the correct priority ranking for tools and provides accurate information for tool invocation. In contrast, the zero-shot result shows that the model fails to select the correct tool.
- In the second example, the GenTool -finetuned LLaMA-8B model selects the most useful tool from two viable options. However, the LLaMA-8B zero-shot result fails to achieve this and instead outputs invalid tool.

These examples illustrate the superiority of GenTool in guiding the model to make better decisions in tool selection and usage.

G Error Analysis

We analyze incorrect examples where LLaMA-8B fails to generate the correct tool call. The errors can be summarized into seven categories, with examples shown in from Table 18 to Table 23 :

1. **Insufficient Query Understanding:** Table 18 shows that the model fails to parse the query correctly, resulting in incorrect parameter inputs.
2. **Parameter Misinterpretation:** Table 19 shows that the model provides parameter values inconsistent with tool specifications.
3. **Query Ambiguity:** Table 20 shows that query ambiguity leads to hallucinated information in the tool call.
4. **Tool Repetition:** Table 21 shows that the model redundantly calls the same tool multiple times, despite needing only one call.

5. **The Second Best Tool Selection:** Table [22](#) shows that the model selects the second-best tool instead of the optimal one.
6. **Default Response Generation:** The model defaults to Generate_Response despite available specialized tools (Table [23](#)).

Toolsets	1. flight_pickup_service Description: Service to arrange pick-up for flights Required parameters: {"flight_number": "Flight number", "arrival_time": "Arrival time", "pickup_location": "Pick-up location", "destination": "Destination"} Returns: {"status": "Pick-up service status", "service_details": "Service details"}
	2. car_transfer_service Description: Tool to arrange car transfer services Required parameters: {"pickup_time": "Pickup time", "pickup_location": "Pickup location", "destination": "Destination", "vehicle_type": "Vehicle type", "passenger_name": "Passenger name"} Returns: {"transfer_status": "Transfer status", "service_details": "Service details", "confirmation_code": "Confirmation code"}
	3. simple_car_rental Description: Basic Car Rental Service Required parameters: {"pickupLocation": "Pickup location", "dropOffLocation": "Drop-off location", "pickupTime": "Pickup time", "dropOffTime": "Drop-off time", "carCode": "Car code"} Returns: {"status": "Rental status", "cost": "Rental cost"}
	4. airport_dropoff_service Description: Service to arrange drop-off to the airport Required parameters: {"dropoff_time": "Drop-off time", "flight_number": "Flight number", "pickup_location": "Pickup location", "airport_name": "Airport name"} Returns: {"status": "Drop-off status", "service_details": "Service details"}
	5. taxi_booking Description: Tool for booking a taxi service Required parameters: {"pickup_time": "Pickup time", "pickup_location": "Pickup location", "destination": "Destination", "passenger_name": "Passenger name", "luggage_count": "Luggage count"} Returns: {"booking_status": "Booking status", "taxi_service_information": "Service info", "confirmation_number": "Confirmation number"}
User query:	I need to know the exact pick-up and drop-off times for a car rental reservation with the code ABC123, including any grace periods for late returns.
Ground truth tool calling:	simple_car_rental {"carCode": "ABC123"}
Predicted tool calling (GenTool):	{"The output of the first task: "["simple_car_rental", "generate_response", "car_transfer_service", "flight_pickup_service", "airport_dropoff_service", "taxi_booking"], "The output of the second task: ": ["simple_car_rental(carCode='ABC123')"]}
Predicted tool calling (Zero-Shot):	car_transfer_service {"pickup_time": "2023-03-15T12:00:00", "pickup_location": "Airport", "destination": "Car Rental Agency", "vehicle_type": "Sedan", "passenger_name": "John Doe"}

Table 16: Two examples predicted by the Llama-3.1-8B-Instruct model, one from finetuning using the GenTool method and the other from performing zero-shot directly. Our method successfully enabled Llama-3.1-8B-Instruct to select the correct tool and rank the tools appropriately.

Toolsets	1. appliance_repair_rescheduling Description: Request to adjust the appliance repair appointment time Required parameters: {"originalRequestID": "Original repair request number", "newTime": "New appointment time"} Returns: {"rescheduling_status": "Status", "newRequestID": "New request number"}
	2. appliance_repair_status_query Description: Appliance Repair Inquiry Tool Required parameters: {"requestID": "Request number", "applianceType": "Appliance type", "applianceBrand": "Brand", "applianceModel": "Model", "contactInformation": "Contact info", "address": "Address"} Returns: {"repair_status": "Inquiry result"}
	3. appliance_repair_cancellation Description: Appliance Repair Cancellation Tool Required parameters: {"requestID": "Request number", "cancellationReason": "Reason"} Returns: {"cancellation_status": "Status"}
	4. appliance_repair_request Description: Appliance Repair Request Tool Required parameters: {"applianceType": "Type", "applianceBrand": "Brand", "applianceModel": "Model", "issueDescription": "Issue", "contactInformation": "Contact", "address": "Address", "time": "Service time"} Returns: {"request_status": "Status", "requestID": "Request ID"}
	5. repair_service_request Description: Request a repair service Required parameters: {"device": "Device type", "make": "Make", "model_number": "Model", "issue": "Issue", "phone": "Phone", "address": "Address", "appointment_time": "Time"} Returns: {"request_status": "Status", "service_id": "Service ID"}
User query:	Please help me schedule a repair service for my LG microwave, model LMV2031ST, which is not heating food. My contact information is 876-543-2109, and my address is 404 Cedar St. I am available on Friday at 1 PM.
Ground truth tool call:	repair_service_request {"device": "microwave", "make": "LG", "model_number": "LMV2031ST", "issue": "not heating food", "phone": "876-543-2109", "address": "404 Cedar St", "appointment_time": "Friday at 1 PM"}
Predicted tool calling (GenTool):	{ "The output of the first task: ": ["repair_service_request", "appliance_repair_request", "generate_response", "appliance_repair_status_query", "appliance_repair_cancellation", "appliance_repair_rescheduling"], "The output of the second task: ": ["repair_service_request(device='microwave', make='LG', model_number='LMV2031ST', issue='not heating food', phone='876-543-2109', address='404 Cedar St', appointment_time='Friday at 1 PM')"] }
Predicted tool calling (Zero-shot):	appliance_repair_status_query {"requestID": "876-543-2109", "applianceType": "LG microwave", "applianceBrand": "LG", "applianceModel": "LMV2031ST", "contactInformation": "876-543-2109", "address": "404 Cedar St."}

Table 17: Two examples were predicted by the Llama-3.1-8B-Instruct model: one using the GenTool method for fine-tuning and the other performing zero-shot inference directly. Our method successfully positioned the two useful tools before “generate_response” and ultimately selected the correct tool.

Toolsets	1. schedule_repair_service Description: Tool to schedule a repair service for home appliances Required parameters: {"appliance": "Type and brand", "model": "Model number", "problem": "Issue description", "contact": "Contact number", "location": "Service location", "appointment_time": "Service time"} Returns: {"status": "Status of the repair service request", "serviceID": "ID of the scheduled service"}
	2. appliance_repair_cancellation Description: Appliance Repair Cancellation Tool Required parameters: {"requestID": "Repair request number", "cancellationReason": "Reason for cancellation"} Returns: {"cancellation_status": "Repair cancellation status"}
	3. appliance_repair_status_query Description: Appliance Repair Inquiry Tool Required parameters: {"requestID": "Request number", "applianceType": "Type of appliance", "applianceBrand": "Brand name", "applianceModel": "Model number", "contactInformation": "Contact info", "address": "Service address"} Returns: {"repair_status": "Repair inquiry result"}
	4. appliance_repair_request Description: Appliance Repair Request Tool Required parameters: {"applianceType": "Type of appliance", "applianceBrand": "Brand name", "applianceModel": "Model number", "issueDescription": "Problem details", "contactInformation": "Contact info", "address": "Service address", "time": "Service time"} Returns: {"request_status": "Repair request status", "requestID": "Repair request ID"}
	5. appliance_repair_rescheduling Description: Request to adjust the appliance repair appointment time Required parameters: {"originalRequestID": "Original request number", "newTime": "New appointment time"} Returns: {"rescheduling_status": "Rescheduling status", "newRequestID": "New repair request number"}
User query:	I need to arrange a repair service for my LG washing machine, model WM3900HBA, which is making a loud noise during the spin cycle. Please schedule it for Saturday at 2 PM. My contact number is 987-654-3210, and my address is 456 Elm St.
Ground truth tool calling:	schedule_repair_service {"appliance": "LG washing machine", "model": "WM3900HBA", "problem": "making a loud noise during the spin cycle", "contact": "987-654-3210", "location": "456 Elm St", "appointment_time": "Saturday at 2 PM"}
Predicted tool calling:	schedule_repair_service {"appliance": "washing machine", "model": "WM3900HBA", "problem": "loud noise", "contact": "987-654-3210", "location": "456 Elm St", "time": "Saturday 2 PM"}

Table 18: LLaMA-3.1-8B-Instruct error case: Here the model lacks a comprehensive understanding of the query. During parsing, it overlooks key textual information. “LG” is a brand and should be associated with “washing machine.”

Toolsets	1. add_reminder Description: Add reminder event Required parameters: {"reminder_id": "ID of the reminder", "event_title": "Event title", "reminder_time": "Time of the reminder", "reminder_location": "Location of the reminder", "reminder_frequency": "Frequency of the reminder (days)", "reminder_name": "Name of the reminder"} Returns: {"status": "Whether the reminder was successfully added"}
	2. create_event_reminder Description: Create a one-time event reminder Required parameters: {"event_name": "Name of the event", "event_time": "Time of the event", "event_location": "Location of the event"} Returns: {"confirmation": "Confirmation of the event reminder creation"}
	3. set_reminder Description: Set a reminder for an event Required parameters: {"event_id": "ID of the event", "reminder_time": "Time before the event to set the reminder"} Returns: {"confirmation": "Confirmation of the reminder being set"}
	4. add_event_to_calendar Description: Add an event to the calendar Required parameters: {"event_name": "Name of the event", "date": "Date of the event", "start_time": "Start time of the event", "end_time": "End time of the event", "location": "Location of the event"} Returns: {"status": "Status of the event addition"}
	5. add_event Description: Add a single event to the schedule Required parameters: {"event_name": "Name of the event", "start_time": "Start time of the event", "end_time": "End time of the event", "location": "Location of the event", "reminder": "Reminder setting (e.g., '15 minutes before')"} Returns: {"status": "Status of the event addition"}
User query:	Could you create a one-time event reminder for a dinner reservation at "The Gourmet Bistro" at 7:00 PM on October 25th, 2023?
Ground truth tool calling:	create_event_reminder {"event_name": "Dinner Reservation at The Gourmet Bistro", "event_time": "2023-10-25T19:00:00", "event_location": "The Gourmet Bistro"}
Predicted tool calling:	create_event_reminder {"event_name": "The Gourmet Bistro", "event_time": "2023-10-25T19:00:00", "event_location": "Dinner Reservation"}

Table 19: LLaMA-3.1-8B-Instruct error case: Parameter type confusion - model places location “gourmet bistro” in event name field instead of location field.

Toolsets	1. real_estate_search_tool Description: Real estate search tool that allows for setting multiple search criteria for filtering parameters: {"location": "Geographic location of the property", "priceRange": "Price range (format: 'min-max')", "areaRange": "Area range (format: 'min-max')", "propertyType": ["apartment", "villa", "townhouse", "condo"], "bedrooms": "Number of bedrooms", "bathrooms": "Number of bathrooms"} Returns: {"listings": [Array of properties with details including id, location, price, area, type, bedrooms, bathrooms, imageURL]}
	2. market_price_checker Description: Market Price Checker Tool Required parameters: {"marketType": ["Energy", "Agriculture", "Metals"], "commodity": "Commodity Name"} Returns: {"current_price": "Current Market Price"}
	3. price_comparison_tool Description: Product price query and comparison tool Required parameters: {"productID1", "productID2", "date1", "date2", "price1", "price2", "storeName1", "storeName2"} Returns: {"price": "Product price", "lowestStore": "Store with lowest price"}
	4. simple_apartment_comparator Description: Tool to compare a specific apartment with the market price of similar properties Required parameters: {"apartmentID": "ID of apartment to compare", "marketPrice": "Market price of similar properties"} Returns: {"comparison": "Comparison result"}
	5. compare_prices Description: Tool to compare prices of two products Required parameters: {"productId1": "First Product ID", "productId2": "Second Product ID"} Returns: {"priceDifference": "Price difference between products"}
User query:	Please compare for me the finely decorated two-bedroom apartments around 100 square meters in the center of Beijing, priced between 3 million and 5 million RMB, especially the one priced at 3.2 million near the subway station, with the market price of other similar properties, so that I can make a purchase decision.
Ground truth tool calling:	real_estate_search_tool {"location": "center of Beijing", "priceRange": "3.2 million", "areaRange": "around 100 square meters", "propertyType": "apartment", "bedrooms": 2, "bathrooms": 1} {"location": "center of Beijing", "priceRange": "3-5 million", "areaRange": "around 100 square meters", "propertyType": "apartment", "bedrooms": 2, "bathrooms": 1}
Predicted tool calling:	simple_apartment_comparator {"apartmentID": "A123", "marketPrice": 3200000}

Table 20: LLaMA-3.1-8B-Instruct error case: Model incorrectly invokes simple_apartment_comparator and hallucinates A123, failing to recognize the query’s intent for comparative property search

Toolsets	1. savings_bond_lookup Description: Lookup tool for savings bonds Required parameters: {"bond_term": "Term of the bond in months", "issue_date": "Date when the bond was issued", "interest_method": "Method of paying interest", "coupon_rate_type": "Type of coupon rate", "term_variability": "Indicates if the term is variable", "keyword": "Keyword for bond search"} Returns: {"bond_info": "Information about the bond"}
	2. job_info_search Description: Job Information Query Tool Required parameters: {"jobTitle": "Job Title", "industry": "Industry Type", "location": "Work Location", "postDate": "Date of Job Posting"} Returns: {"job_descriptions": "Job Descriptions Summary", "companyName": "Company Name", "jobDescription": "Job Requirements", "salaryRange": "Salary Range"}
	3. train_ticket_query Description: Train Ticket Booking Inquiry Tool Required parameters: {"departureDate": "Departure Time", "startStation": "Departure Station", "endStation": "Arrival Station", "trainID": "Train Number", "seatType": "Seat Type"} Returns: {"ticket_status": "Booking List", "order_details": "Order Details"}
	4. company_credit_check Description: Enterprise Credit Inquiry Tool Required parameters: {"companyName": "Company name", "country": "Company location", "checkDate": "Inquiry date", "checkType": ["financial", "legal", "business"]} Returns: {"credit_score": "Enterprise credit score", "credit_rating": "Credit rating", "credit_status": "Credit status"}
	5. basic_job_search_tool Description: A simple tool for searching job positions based on basic criteria Required parameters: {"jobTitle": "Title", "location": "Location", "salary": "Salary range"} Returns: {"jobListings": [Array of jobs with company, title, location, salary]}
User query:	I need to find the savings government bonds issued from today with a 36-month term, periodic interest payments, fixed coupon rate, and a variable term type, and view the details of the first record, especially using 'government bonds' as the keyword to obtain relevant information.
Ground truth tool calling:	savings_bond_lookup {"bond_term": 36, "issue_date": "today's date", "interest_method": "periodic", "coupon_rate_type": "fixed", "term_variability": "variable", "keyword": "government bonds"}
Predicted tool calling:	savings_bond_lookup {"bond_term": 36, "issue_date": "today's date", "interest_method": "periodic interest payments", "coupon_rate_type": "fixed coupon rate", "term_variability": "variable term type", "keyword": "government bonds"} savings_bond_lookup {"bond_term": 36, "issue_date": "today's date", "interest_method": "periodic interest payments", "coupon_rate_type": "fixed coupon rate", "term_variability": "variable term type", "keyword": "government bonds"}

Table 21: LLaMA-3.1-8B-Instruct error case: Redundant duplicate tool calls in response output.

Toolsets	1. part_time_job_tool Description: Part-time job information collection and management tool, only applicable to college students Required parameters: {"jobType": ["Education", "Sales", "Media", "Service Industry", "IT"], "hourlyWage": "number >= 1", "workingHours": "format 'HH:MM-HH:MM'", "jobLocation": "string", "experienceRequired": "boolean", "studentIdentityRequired": "boolean", "jobDescription": "string", "contact": "string", "postDate": "date"} Returns: {"success": "boolean", "jobId": "string"}
	2. job_info_search Description: Job Information Query Tool Required parameters: {"jobTitle": "Job Title", "industry": "Industry Type", "location": "Work Location", "postDate": "Date of Job Posting"} Returns: {"job_descriptions": "Summary", "companyName": "Company", "jobDescription": "Requirements", "salaryRange": "Salary"}
	3. basic_job_search_tool Description: A simple tool for searching job positions based on basic criteria Required parameters: {"jobTitle": "Title", "location": "Location", "salary": "Salary range"} Returns: {"jobListings": [Array of jobs with company, title, location, salary]}
	4. job_hiring_tool Description: Job hiring tool for posting vacancies and filtering candidates Required parameters: {"jobPosition": "string", "educationRequirement": ["Doctorate", "Master's", "Bachelor's", "Associate", "Other"], "workExperience": "string", "salaryRange": "string", "jobLocation": "string", "companySize": "string", "requiredSkills": "comma-separated skills"} Returns: {"candidates": [Array of candidates], "jobPosted": "boolean"}
	5. savings_bond_lookup Description: Lookup tool for savings bonds Required parameters: {"bond_term": "Term of the bond in months", "issue_date": "Date when the bond was issued", "interest_method": "Method of paying interest", "coupon_rate_type": "Type of coupon rate", "term_variability": "Indicates if the term is variable", "keyword": "Keyword for bond search"} Returns: {"bond_info": "Information about the bond"}
User query:	Can you find job listings for a Content Writer in Sydney with a salary range of 50k-70k?
Ground truth tool calling:	basic_job_search_tools {"jobTitle": "Content Writer", "location": "Sydney", "salary": "50k-70k"}
Predicted tool calling:	job_info_search {"jobTitle": "Content Writer", "location": "Sydney", "salaryRange": "50k-70k"}

Table 22: LLaMA-3.1-8B-Instruct error case: For this simple query, the model should have called the “basic_job_search_tool”. However, it incorrectly chose the “job_info_search” tool, and the parameters it used for calling this tool were inconsistent with the tool’s descriptions.

Toolsets	1. add_event_to_calendar Description: Add an event to the calendar Required parameters: {"event_name": "Event name", "date": "Event date", "start_time": "Start time", "end_time": "End time", "location": "Event location"} Returns: {"status": "Event addition status"}
	2. create_event Description: Create a new event in the calendar Required parameters: {"event_title": "Event title", "date": "Event date", "start_time": "Start time", "end_time": "End time", "venue": "Event venue"} Returns: {"status": "Event creation status"}
	3. create_event_label Description: Create a label for an event Required parameters: {"event_id": "Event ID", "event_name": "Event name", "label_color": "Label color", "label_description": "Label description"} Returns: {"label_creation_status": "Label creation status"}
	4. add_event Description: Add a new event to the calendar Required parameters: {"event_name": "Event name", "event_date": "Event date", "event_time": "Event time"} Returns: {"status": "Event addition status"}
	5. delete_event Description: Delete an event from the calendar Required parameters: {"event_id": "Event ID", "event_name": "Event name" } Returns: {"status": "Event deletion status" }
User query:	Please set a one-time reminder for me to attend the 'Team Building Activity' held in the company meeting room at 2 PM three days from now, and set a recurring reminder to check the project progress every Monday morning at 8 AM in my office.
Ground truth tool calling:	create_event {"event_title": "Team Building Activity", "date": "2023-10-14", "start_time": "14:00", "end_time": "16:00", "venue": "company meeting room"} create_event {"event_title": "Check Project Progress", "date": "2023-10-16", "start_time": "08:00", "end_time": "09:00", "venue": "office"}
Predicted tool calling:	generate_response()

Table 23: LLaMA-3.1-8B-Instruct error case: Model defaults to generate_response despite availability of suitable create_event tool.