GENESISFUNC: Multi-Agent Data Generation for Accurate and Generalizable Function-Calling

Hao-Xiang Xu[1], **Chong Deng**[2], **Jiaqing Liu**[2], **Wen Wang**[2], **Qian Chen**[2], **Lujia Bao**[2], **Xiangang Li**[2], **Zhen-Hua Ling**[1*]
[1]University of Science and Technology of China, Hefei, Anhui, China
[2]Independent Researcher
nh2001620@mail.ustc.edu.cn

Abstract

Large Language Models (LLMs) extend their capabilities through function-calling (FC), which relies on training data with high quality, diversity, and broad coverage of scenarios. However, obtaining and annotating real function-calling data is challenging, while synthetic data from existing pipelines often suffers from unreliable APIs, limited tool scalability, insufficient diversity, and weak quality control. To address these, we present GENESISFUNC, an automated pipeline for generating FC training data. Starting from reliable tools in widely used public benchmarks, our GENESISFUNC employs a multi-agent framework to support a dialogue generation system that produces conversations spanning diverse scenarios, while maintaining both diversity and quality throughout the process. The accuracy of the data is further reinforced through a multi-stage evaluation system. We fine-tune an 8B LLM on the synthetic dataset and show through extensive experiments that it outperforms similarly sized open-source models in in-domain FC performance and out-of-domain generalization, while reaching FC capabilities comparable to some of the latest API-based models. In addition, our method demonstrates strong potential to scale effectively across downstream tools, underscoring its real-world applicability. The complete pipeline and the constructed dataset is available at https://github.com/famoustourist/GenesisFunc.

1 Introduction

Tool learning represents a crucial step in advancing the frontier of Large Language Models (LLMs), as it transforms them from passive language processors into proactive agents capable of interacting with dynamic environments (Huang et al., 2024; Qin et al., 2024). By bridging internal reasoning with external execution, tool-equipped LLMs are no longer constrained by static training data but can generalize to open-ended tasks and adapt to evolving user needs (Patil et al., 2024; Qu et al., 2025). This paradigm thus underscores not only the practical value of enhancing task coverage across diverse applications such as workflow automation and travel planning (Zhong et al., 2024; Hao et al., 2024), but also the theoretical importance of extending the fundamental boundaries of LLM capabilities (Liu et al., 2024, 2025a).

Despite the rapid progress in tool learning, the field still faces fundamental challenges that constrain its broader applicability. The effectiveness of function-calling (FC) relies heavily on high-quality training data, yet collecting and annotating real-world data is costly and labor-intensive (Qin et al., 2024). Moreover, real-world function-calls are inherently complex and diverse, often characterized by ambiguous user intents, rapidly changing dynamic environments, multi-task requirements, and extended multi-turn interactions (Tang et al., 2023; Patil et al., 2024; Liu et al., 2024; Abdelaziz et al., 2024). These intertwined challenges underscore the urgent need for more robust data generation pipelines that can produce *accurate*, *diverse*, and *broadly representative training data* to support realistic function-calling scenarios.

Increasing research efforts have sought to design automated pipelines for synthesizing training data, which are then used to build tool-augmented LLMs (Qu et al., 2025; Liu et al., 2025a). While progress has been made, fundamental limitations remain. Existing methods often rely on publicly available or manually constructed APIs, which are frequently unreliable and difficult to scale across broader tool sets, thereby constraining function-calling capabilities. In addition, the generated data often suffers from insufficient diversity and inadequate quality control, further weakening performance. From the perspective of

*Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28594–28616

scenario coverage, most approaches still emphasize single-turn or isolated function calls, leaving them unable to capture more realistic settings such as multi-turn dialogues or multi-task interactions. Together, these limitations suggest the insufficiency of current approaches and highlight the need for more effective and scalable solutions.

In this paper, we introduce **GENESISFUNC**, a three-stage automated pipeline designed to generate function-calling training data that is **high-quality**, **diverse**, and **broadly representative of real-world scenarios**. To ensure the reliability of APIs and enable seamless extension to a wider range of tools, GENESISFUNC begins by **selecting a curated seed set** from the widely adopted BFCL benchmark (Patil et al., 2025). These APIs are verified to cover multiple domains, making them not only dependable but also practical for extension to downstream tools, since such functions are readily accessible in real applications (Zhang et al., 2024). Next, to improve dialogue quality, diversity, and coverage, GENESISFUNC incorporates a **multi-agent-assisted dialogue generation system**. Through the coordinated interaction of agents, the framework leverages history-aware role differentiation and parameter-slot selection to expand diversity, while an agent-based scoring mechanism safeguards the quality of synthesized dialogues. Finally, to further guarantee correctness, GENESISFUNC employs a **multi-stage evaluation module** that combines rule-based and model-based checks, followed by targeted human review, thereby ensuring both conformity and executability of the training data. Through this end-to-end process, GENESISFUNC produces robust, diverse, and scenario-rich training data, which significantly enhances the function-calling capabilities of LLMs.

To validate the effectiveness of GENESISFUNC, we conduct supervised fine-tuning on Qwen3-8B (Yang et al., 2025) using the training data generated by our pipeline and evaluate the zero-shot performance of the fine-tuned model across several public benchmarks. Since the tools are sourced from BFCL (Patil et al., 2025), we report results not only on the in-domain BFCL benchmark but also on the out-of-domain API-Bank (Li et al., 2023) and ACEBench (Chen et al., 2025a), thereby assessing both in-domain performance and generalization to unseen domains. Under a consistent evaluation protocol, our model consistently surpasses open-source baselines of comparable scales and remains highly competitive with API-based models (OpenAI, 2023). Furthermore, we demonstrate that our approach can scale up to more downstream tools and verify that the data generated by our pipeline can also be used through reinforcement learning to enhance function-calling performance in multi-turn dialogue scenarios.

In summary, this paper makes three primary contributions: (1) We introduce GENESISFUNC, an automated pipeline for generating high-quality function-calling training data for LLMs. The pipeline begins with selecting reliable APIs and integrates a **Multi-agent Dialogue Generation Module** and a **Multi-stage Evaluation Module** to ensure robustness, diversity, and scenario coverage. (2) Through extensive verification, GENESISFUNC produces datasets that are **accurate and diverse while covering real-world scenarios**. Moreover, our approach demonstrates **strong generalizability to downstream tools**, underscoring its practical applicability. (3) We fine-tune Qwen3-8B with training data generated by our pipeline and evaluate the GENESISFUNC-8B on three widely used benchmarks, BFCL (Patil et al., 2025), API-Bank (Li et al., 2023), and ACEBench (Chen et al., 2025a). GENESISFUNC-8B consistently outperforms open-source LLMs of comparable scales and remains highly competitive with API-based models.

2 Related Work

LLM Function-Calling Paradigms. Equipping LLMs with executable tools enables reliable and specialized problem solving (Qin et al., 2024). Existing approaches mainly follow two paradigms: prompting and fine-tuning. Prompting leverages in-context tool specifications and demonstrations (Mialon et al., 2023; Hsieh et al., 2023). ReAct (Yao et al., 2023) combines reasoning with API calls for multi-step tasks, but its performance is constrained by pretraining and degrades with increased tool complexity. These limitations motivate fine-tuning via supervised or reinforcement learning (Tang et al., 2023; Abdelaziz et al., 2024). Representative methods include ToolACE (Liu et al., 2025a), which constructs large-scale tool-use corpora automatically, and RL-based approaches such as ToolRL (Qian et al., 2025) and AWPO (Lin et al., 2025), which enhance function-calling through policy optimization with reasoning rewards.

Data Synthesis for Function-Calling. As LLMs advance, reliance solely on human-authored corpora becomes insufficient for sustained

progress (Bauer et al., 2024). To expand supervision without heavy annotation costs, recent works adopt prompt-driven transformations to augment existing datasets. For example, Yu et al. (2024) proposes targeted prompting to elicit rare skills and long-tail behaviors from base models. However, purpose-driven data remain limited in tool-use scenarios. Prior efforts adapt resources from adjacent domains (Basu et al., 2024) or synthesize samples around public APIs (Liu et al., 2024). Moreover, ToolACE (Liu et al., 2025a) constructs large-scale function-calling corpora via automated pipelines, while ToolForge (Chen et al., 2025b) introduces automated tool-use synthesis with reduced reliance on real APIs.

Our work is distinguished from the most relevant studies (Qin et al., 2024; Liu et al., 2024, 2025a) in the following aspects. Prior works often rely on annotated or synthetic APIs, which lack reliability and struggle to scale across larger tool sets. These approaches also face limitations in diversity, quality, and coverage. In contrast, GENESISFUNC is built upon reliable tools drawn from public benchmarks and offers stronger scalability. Furthermore, by leveraging a multi-agent dialogue generation framework and a multi-stage verification system, GENESISFUNC produces tool-use training data with richer diversity, higher quality, and broader scenario coverage than prior works.

3 Preliminary

Given a user query q and a candidate tool set $\mathcal{T} = \{t_1, \ldots, t_n\}$, where each tool t_i is defined by a name, a usage description, and a schema specifying required and optional parameters, the goal is to generate a valid sequence of tool calls by selecting tools and filling arguments with appropriate values and units. This process can be framed as follows:

$$\mathcal{S} = \big[t_1(a_1), \ldots, t_k(a_k)\big] = g_\phi\big(q, \mathcal{T}\big), \quad (1)$$

where $g_\phi(\cdot)$ denotes an LLM with parameters ϕ, k is the number of invocations, and a_i denotes the argument payload for the i-th call ($1 \le i \le k$), which is a set of parameter-value pairs, that is, $a_i = [\, r_1 : w_1,\ r_2 : w_2, \ldots, r_\ell : w_\ell \,]$, with parameter names r_j and corresponding values w_j. The query q may be a single turn or a full multi-turn history.

For fine-tuning, the pair $(q, \mathcal{T})$ is treated as input context, while the gold-standard tool-call sequence $\mathcal{S}$ serves as the supervised target. Formally, the training samples can be represented as $\{\, \langle q, \mathcal{T} \rangle, \mathcal{S} \,\}$, where each training instance connects a user query and the candidate tools with the corresponding sequence of tool invocations.

4 Data Generation Pipeline

Synthetic data plays a vital role in enhancing the function-calling capability of LLMs (Liu et al., 2025a). We propose an automated pipeline GENESISFUNC that employs a structured, three-stage process to generate high-quality training data for tool-augmented LLMs, as illustrated in Figure 1. First, reliable and functionally diverse tools are selected from public datasets and placed into the **Tool Pool**. Next, the **Multi-Agent Dialogue Generation** module leverages a multi-agent-assisted framework to produce tool-use dialogues that are diverse, accurate, and broadly representative. Finally, the **Multi-Stage Evaluation** module systematically examines the correctness of the generated dialogues to guarantee training data quality.

4.1 Tool Pool

The quality of training data for enhancing LLM tool-use abilities depends critically on the reliability and coverage of the underlying APIs. However, existing datasets often rely on manually annotated APIs or handcrafted synthetic pipelines, which are either costly to scale or difficult to extend to real-world tools, limiting model generalizability.

To address these issues, GENESISFUNC constructs a curated Tool Pool entirely sourced from the BFCL evaluation set (Patil et al., 2025). The pool is built through a two-stage filtering process: GPT-4o (OpenAI, 2023) is first used to perform semantic clustering over all tools to remove redundant or highly similar ones, followed by light human verification to ensure correctness and usability. This process yields a curated Tool Pool of 1,000 tools. Since tools in BFCL are manually designed to cover a broad range of real-world usage scenarios, we further verify that the curated pool maintains high functional diversity across multiple domains. By relying on widely used and practically accessible tools, **the Tool Pool balances reliability, diversity, and scalability, and serves as a critical foundation for the subsequent data generation modules in GENESISFUNC**.

4.2 Multi-Agent Dialogue Generation

For LLMs, the quality of fine-tuning data significantly affects downstream performance (Liu

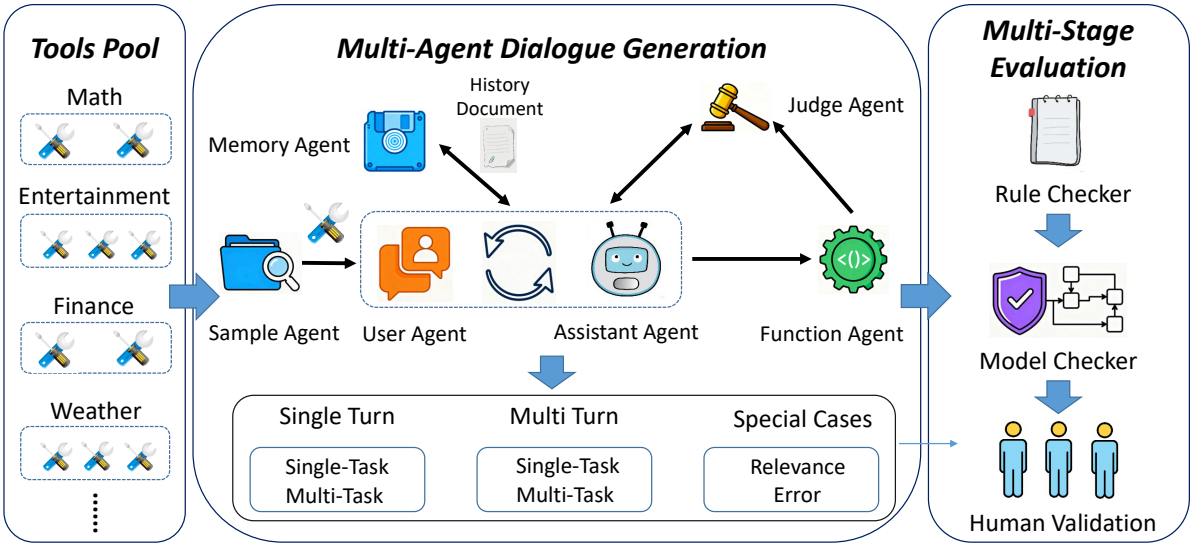


Figure 1: The overall framework of GENESISFUNC, consisting of a **Tool Pool** built from reliable tools in open-source benchmarks, a **Multi-Agent Dialogue Generation** system, and a **Multi-Stage Evaluation** process.

et al., 2025b). We propose a multi-agent dialogue generation module that leverages a multi-agent framework to assist the dialogue generation system in synthesizing function-calling dialogues. By leveraging agent interaction and collaboration, this module produces dialogues that are high-quality, diverse, and broadly representative, which is crucial to enhance tool-use capabilities of LLMs.

4.2.1 Multi-Agent Framework

The multi-agent framework includes four LLM-based agents: *Sample Agent*, *Memory Agent*, *Function Agent*, and *Judge Agent*. We adopt Gemini-2.5 Pro (Comanici et al., 2025) as the backbone of all agents, which is an engineering choice due to Gemini-2.5 Pro's *stable API-following performance* rather than an algorithmic requirement. Importantly, the multi-agent framework is *model-agnostic*; Appendix A.1 shows that **replacing proprietary models with strong open-source alternatives retains the overall workflow and leads to only moderate data quality degradation and slight downstream performance drops**. System prompts and detailed agent descriptions and analyses are also in Appendix A.1.

Sample Agent selects a subset $\mathcal{T}_s \subseteq \mathcal{T}$ from the global Tool Pool, including target tools $\mathcal{T}_{\text{target}}$ and distractors $\mathcal{T}_{\text{dist}}$. Target tools are grouped by semantic or functional similarity, while distractors are chosen with varied relevance to increase realism and require the model to distinguish between strongly and weakly related tools.

Memory Agent records dialogue instances $\mathcal{D} = \{d_1, \ldots, d_k\}$ and assigns each a type label $\tau(d_i) \in \mathcal{C}$ reflecting the semantic context of tool usage, where the same math tool may address architectural geometry in one case and year calculations in another. It then summarizes past pairs $(d_i, \tau(d_i))$ and guides the generator to produce new dialogues with unseen τ, improving semantic diversity.

Function Agent selects one or more tools $\{t_1, \ldots, t_k\} \subseteq \mathcal{T}_s$ to address the user query q, and extracts both required and optional parameters for each tool t_i. To enhance diversity and realism, a slot-selection strategy is applied: during parameter extraction, the agent randomly chooses a subset $\mathcal{P}' \subseteq \mathcal{P}$ of optional parameters to instantiate, resulting in varied and realistic tool calls.

Judge Agent ensures rigorous quality control by selecting the best dialogue d^* from N candidates $\{d_1, \ldots, d_N\}$ (default $N = 4$) each generated round. Evaluation considers **problem significance** and **tool appropriateness**, with **positional bias controlled** as discussed in Appendix A.1. $N = 4$ balances selection quality and generation efficiency. The selected d^* serves as the final output.

4.2.2 Dialogue Generation System

We design a dialogue generation system to produce three types of function-calling dialogues: **single-turn**, **multi-turn**, and **special-case**. The first two types cover both single-task and multi-task settings, enabling the simulation of simple requests and more complex interaction flows. The special cases, on the other hand, are designed to cover scenarios for relevance checking and error detection. The

overall system is composed of two agents, a user agent U and an assistant agent A, both instantiated with Gemini-2.5 Pro, that interact with each other to generate task-oriented conversations.

The detailed algorithm of the multi-agent dialogue generation module and dialogue examples are provided in Appendix A.2. The dialogue generation system grounds each conversation in the sampled tool set $\mathcal{T}_s$ drawn from the curated pool. Every dialogue contains user requests that can be handled by $\mathcal{T}_s$ and tool call answers that explicitly specify the selected tools and the full argument payload. The assistant's action space is $\mathcal{A}_{\text{asst}} = \{\texttt{call}, \texttt{ask}, \texttt{answer}\}$. The assistant can issue a tool call $t(a)$, request clarification, summarize tool outputs, or provide a direct non-tool reply when a call is unnecessary. The system records the chosen tool t and the payload a in the turn-level state and appends them to the *turn-level trajectory* for later evaluation and reuse. These recorded trajectories, after being categorized by the memory agent, are leveraged to guide subsequent dialogue generation, preventing the system from reproducing similar contexts and encouraging scenario diversity.

At turn t, the user agent U issues a request q_t based on the tool set $\mathcal{T}_s$. The assistant agent A receives the request q_t together with the dialogue history $\mathcal{D}$, follows the system prompts, and generates the next action. In single-turn dialogues, A constructs a problem solvable with $\mathcal{T}_s$, produces a tool call, and returns the final answer. In multi-turn dialogues, A alternates with U by requesting additional information when constraints are missing, and once the target length is reached or a stop signal triggers, the model outputs either `call` or `answer`. In special cases, prompts guide the generation of unsolved or erroneous samples, such as mismatched tools or invalid parameter values, to support relevance checking and error detection.

4.3 Multi-Stage Evaluation

We introduce a multi-stage evaluation system to assess the quality of synthesized dialogues, since inaccurate training data may significantly weaken the models' ability to understand and execute functions. This system integrates a rule-based checker and a model-based checker, with final verification by human experts to ensure the accuracy of the resulting training data. More details can be found in Appendix A.3.

(1) Rule Checker. Without executing tools, the Rule Checker performs basic compliance checks on synthesized dialogues to quickly filter out samples with format and alignment issues. It validates four aspects: completeness of the tool definition, compliance of the call format and parameters, soundness of the dialogue structure, and consistency between tool results and the assistant's statements (rule details in Appendix A.3) Each rule returns a *pass* or *fail* flag with a hint for correction, and outputs are aggregated for the downstream Model Checker and Human Validation.

(2) Model Checker. After format screening, the Model Checker employs GPT-4o (OpenAI, 2023) to evaluate semantic quality and task completion beyond the static rules (the prompt is detailed in Appendix A.3). Unlike the Judge Agent that ranks candidate dialogues *during* generation, the Model Checker works *post-hoc* to verify finalized dialogues. Given the dialogue context and tool calls, it checks faithfulness, task satisfaction, and compliance, returning a rationale and a confidence score. Only samples with confidence scores above the threshold ($\theta = 0.75$) are retained.

(3) Human Validation. After the Rule Checker and the Model Checker complete automated screening, the remaining error rate is **below 5%**. Manual analysis shows that **over 80%** of the errors after automated screening stem from parameter extraction rather than function selection or intent understanding. Samples that fail the Rule Checker or obtain low-confidence from the Model Checker are routed to a human review, with higher priority given to errors that prevent correct function execution and errors in high-impact functions that are commonly used in core real-world scenarios. These samples account for about **5%** of all samples. Human experts review each of these samples and provide revisions; the approved or revised dialogues are added to the training data after passing the second-pass human validation. Overall, the Human Validation stage only costs $\sim$**15 human-hours**.

Overall comparison with other strong synthetic function-calling datasets. Appendix B provides detailed analyses of quantity, quality, coverage, diversity of the final training data. Compared to other strong synthetic function-calling datasets, our dataset achieves higher quality through richer multi-tool and multi-turn compositions, broader coverage across diverse scenarios, and balanced parameter utilization patterns, while maintaining efficient construction by reusing existing tool schemas and avoiding additional manual designs.

Table 1: **Accuracy** on the **In-Domain** BFCL dataset. GENESISFUNC-8B is fine-tuned on Qwen3-8B using training data generated by our pipeline. The best results in each category are in **bold** and the second best results are underlined. We report mean and standard deviation (±SD) of the *Overall* accuracy based on three independent runs.

Models	Non-Live					Live				
	Simple	*Multiple*	*Parallel*	*Parallel Multiple*	*Overall*	*Simple*	*Multiple*	*Parallel*	*Parallel Multiple*	*Overall*
GPT-4o	76.50	91.00	90.00	78.00	83.88	70.54	70.75	62.50	66.67	70.54
GPT-4o-mini	80.33	92.00	89.50	90.50	88.08	79.46	76.26	87.50	70.83	76.91
Gemini-2.5-Pro	78.67	94.00	93.50	92.00	89.54	85.66	74.36	87.50	83.33	76.83
LLaMA-3.1-8B-Instruct	71.00	95.00	87.50	82.50	84.00	72.87	71.13	50.00	45.83	59.96
LLaMA-3.1-70B-Instruct	78.33	97.00	95.50	**94.00**	91.21	83.33	75.59	87.50	58.33	76.91
xLAM-8B	73.83	93.50	87.50	83.50	84.58	75.58	66.57	56.25	54.17	67.95
ToolACE-8B	81.17	96.00	94.00	93.00	91.04	82.95	79.58	75.25	**85.12**	80.73
DeepSeek-V3	76.58	94.50	92.00	92.00	87.77	86.05	78.82	75.00	66.67	79.94
Qwen3-8B	78.92	95.00	91.50	89.00	88.60	80.23	77.21	81.25	75.00	77.79
Qwen3-32B	79.58	95.00	92.00	93.00	89.90	84.50	80.44	**93.75**	66.67	81.13
Hammer2.1-7B	77.17	96.00	93.00	87.50	88.42	77.13	77.59	87.50	70.83	77.50
Qwen-ToolRL-8B	83.50	89.50	91.25	91.85	89.03	81.48	77.82	72.75	81.50	78.39
GENESISFUNC-8B (Ours)	**86.85**	**97.25**	**96.15**	93.00	**93.31 ± 0.42**	**88.25**	**80.50**	81.50	84.88	**83.78 ± 0.37**

5 Experiments

5.1 Experimental Setup

We fine-tune LLMs on the training data produced by our pipeline and evaluate the resulting models in a broad range of settings. Unless otherwise noted, we conduct SFT (Hu et al., 2022) on Qwen3-8B (Yang et al., 2025) using our training data and denote the resulting model by GENESISFUNC-8B. We compare GENESISFUNC-8B against top-tier API-based and open-source foundation LLMs, as well as representative function-calling models trained with specialized function-calling training data including ToolACE-8B (fine-tuning LLaMA3.1-8B-Instruct on ToolACE-generated data), Hammer2.1-7B (based on Qwen 2.5 coder series), and Qwen-ToolRL-8B (fine-tuning Qwen3-8B with ToolRL datasets). Evaluations are conducted on three commonly used benchmarks: BFCL (Patil et al., 2025), API-Bank (Li et al., 2023), and ACEBench (Chen et al., 2025a), with all results averaged over three independent runs. More experimental details, including evaluation metric definitions and settings, are in Appendix C.

5.2 Main Results

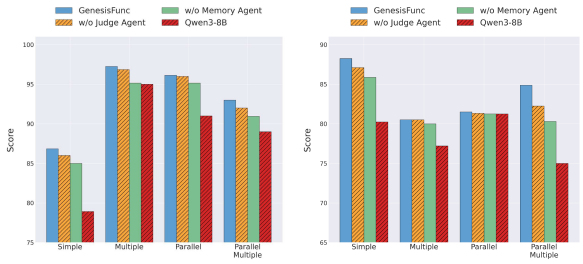
To comprehensively assess function-calling performance, we compare GENESISFUNC-8B with baselines on both in-domain and out-of-domain datasets. The in-domain evaluation measures performance on data aligned with the training distribution, while the out-of-domain evaluation assesses its generalizability to unseen scenarios.

In-Domain Evaluation. Since the tools in pool are drawn directly from BFCL, we treat BFCL as the in-domain benchmark. As shown in Table 1, GENESISFUNC-8B achieves 93.31 on the Non-Live split and 83.78 on the Live split, showing strong and robust performance, and outperforming the API-based models. GENESISFUNC-8B achieves substantial gains over open-source models of similar scale and on some subsets matches or outperforms much larger models such as Qwen3-32B. Compared with the prior open-source function-calling SOTA ToolACE-8B, our model improves overall accuracy by **2.5% relative** on Non-Live and **3.8% relative** on Live. Overall, these results indicate that, **for function-calling, strong and systematic alignment between training data distribution and real tool semantics can significantly narrow the performance gap between smaller models and much larger ones**.

Out-of-Domain Evaluation. To evaluate the generalizability of our fine-tuned model, we conduct experiments on two out-of-domain benchmarks: API-Bank and ACEBench. As shown in Table 2, API-based models maintain a clear advantage over open-source ones, with GPT-4o reaching 85.10 on ACEBench. Open-source models fine-tuned for function calling achieve competitive performance but fall short. Compared with the prior open-source SOTA, our model achieves 64.79 on API-Bank and 78.64 (mean of Normal Overall and

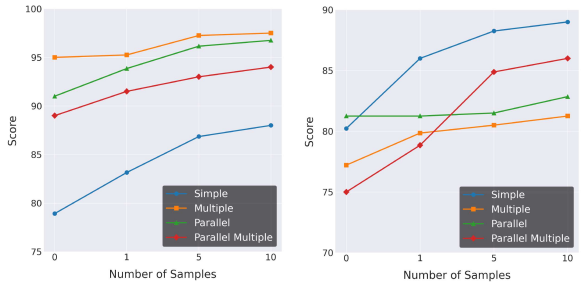
Table 2: **Accuracy** on the **Out-of-Domain** API-Bank and ACEBench datasets. The best results in each category are in **bold** and the second best results are underlined. We report mean and standard deviation (±SD) of the *Overall* accuracy from our model based on three independent runs.

	API-Bank				ACEBench						
					Normal						Special
Models	Level-1	Level-2	Level-3	Overall	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Overall	Overall
GPT-4o	76.19	42.96	35.21	51.45	**90.00**	**78.00**	**77.00**	**85.00**	**83.00**	**82.60**	**87.60**
GPT-4o-mini	74.69	45.93	40.77	53.80	84.33	73.50	66.50	77.00	78.00	75.87	79.90
Gemini-1.5-Pro	75.43	43.26	41.51	53.40	84.50	76.80	64.50	80.00	78.00	76.76	79.00
Qwen3-8B	71.68	52.24	42.75	55.56	80.00	65.50	51.00	68.00	60.00	64.90	76.67
Qwen-ToolRL-8B	73.07	60.81	47.21	60.36	79.00	68.50	52.00	66.00	60.00	65.10	78.67
LLaMA-3.1-8B-Instruct	71.18	37.04	35.88	48.03	51.90	39.80	28.00	66.00	46.00	46.34	46.60
ToolACE-8B	75.94	47.41	45.27	56.21	84.00	74.50	61.00	74.00	58.00	70.30	1.00
GENESISFUNC-8B (Ours)	**79.17**	**64.09**	**51.11**	**64.79 ± 0.41**	88.00	77.00	65.00	74.00	64.00	73.60 ± 0.32	83.67 ± 0.35



(a) Non-Live (b) Live

Figure 2: Ablation study of the proposed agents. We separately remove the Judge Agent and the Memory Agent, and evaluate our GENESISFUNC-8B on BFCL in (a) Non-Live and (b) Live settings.



(a) Non-Live (b) Live

Figure 3: Ablation study on the number of samples. We control the number of dialogues generated per run and evaluate on BFCL in (a) Non-Live and (b) Live settings.

Special Overall) on ACEBench, yielding 7.3% and 9.4% relative gains. Notably, GENESISFUNC-8B performs on par with GPT-4o-mini and Gemini-1.5-Pro. These results validate that **the training data generated by our pipeline effectively enables model generalization to unseen scenarios**.

General Abilities. We also find that **fine-tuning on the training data generated by our pipeline preserves the models general abilities while substantially improving function-calling ability** (more details in Appendix D.1).

5.3 Ablation Study

Ablation on Multi-Agent Framework. Our multi-agent framework improves function-calling ability through agent collaboration, where the Memory Agent enhances dialogue diversity and the Judge Agent ensures accuracy. We conduct ablation studies by removing either agent and fine-tuning Qwen3-8B with LoRA, noting that the Sample Agent and Function Agent are indispensable to the workflow. As shown in Figure 2, BFCL results indicate that removing the Judge Agent causes a clear performance drop, while removing the Memory Agent leads to significant degradation, highlighting the importance of accuracy and diversity and validating the effectiveness of both agents. We further compare alternative slot selection strategies in the Function Agent in Appendix D.2.

Impact of the Number of Samples. To assess how the number of generated samples influences function-calling ability, we use the same set of tools and only vary how many dialogues are produced for each tool. Besides the default setting of 5 dialogues per tool, we also construct datasets with 1 and 10 dialogues per tool. We fine-tune Qwen3-8B on these datasets and report BFCL results in Figure 3. Overall accuracy steadily consistently rises with more samples per tool, with a substantial gain from 1 to 5 samples but only modest improvement from 5 to 10, indicating eventually diminishing returns once sufficient scenario diversity is reached.

Ablation on Multi-Stage Evaluation. Detailed results are in Appendix D.3. We find that models trained on data with the multi-stage evaluation module achieve higher accuracy across all conditions than those trained on non-evaluated data, verifying the effectiveness of this module.

Table 3: **Accuracy** on ACEBench using **reinforcement learning**. The best overall results in each category are marked in bold. The second best results are underlined.

	Normal						Special
Models	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Overall	Overall
Qwen3-8B	80.00	65.50	51.00	68.00	60.00	64.90	76.67
GENESISFUNC-8B	88.00	77.00	65.00	74.00	64.00	73.60	**83.67**
GENESISFUNC-8B-RL(all)	87.00	78.00	66.00	72.00	62.00	73.00	82.45
GENESISFUNC-8B-RL(part)	88.00	79.00	70.00	75.00	64.00	**75.20**	82.88

Table 4: **Overall Accuracy** on BFCL, API-Bank and ACEBench adding the tools in ACEBench. The best overall results in each category are marked in bold. The second best results are underlined.

Models	**BFCL**	**API-Bank**	**ACEBench**
Qwen3-8B	83.20	55.56	70.79
ToolACE-8B	85.89	56.21	35.65
GENESISFUNC	**88.55**	64.79	78.64
+ ACEBench	87.89	**65.11**	**81.87**

5.4 Scalability and Reinforcement Learning

Scalable to More Tools. To assess scalability of our pipeline to more downstream tools, we add tools defined in ACEBench into the pool and use our pipeline to generate high-quality training data for fine-tuning. As shown in Table 4, performance on ACEBench improves markedly from 78.64 to 81.87 after adding these tools, due to better tool alignment. Meanwhile, aggregate results on BFCL remain comparable at 87.89 versus 88.55, and on API-Bank at 65.11 versus 64.79, indicating that introducing additional tool definitions does not notably degrade performance on other benchmarks. Overall, **our pipeline scales well to broader tool inventories and yields consistent gains on targeted benchmarks with no observable degradations on other datasets, across different model sizes and different backbone architectures**, as shown in Appendix D.4 and Appendix D.5.

Enhancing Function-Calling Ability in Multi-Turn Dialogues via RL. Inspired by prior works (Qian et al., 2025; Wan et al., 2025), one research question is *whether applying reinforcement learning (RL) on our data can further enhance the models function-calling capability, particularly in multi-turn dialogue scenarios where performance is less satisfactory*. We conduct two sets of experiments: GENESISFUNC-8B-RL(all) applies RL instead of SFT to Qwen3-8B using our training data; GENESISFUNC-8B-RL(part) focuses on data in multi-turn scenarios, where the model is first SFT-trained on single-turn and special-case data and then RL-trained with multi-turn dialogues. To encourage deeper reasoning during training, we enable the built-in "thinking mode" of Qwen3-8B and augment the training samples with explicit reasoning traces. We use GRPO (Shao et al., 2024) for RL, with rewards designed around two dimensions: format compliance and functional correctness. More details are in Appendix E.

Table 3 compares the performance of different training strategies on ACEBench. The model SFT-ed on our data achieves 83.67. Compared with the default SFT setup, GENESISFUNC-8B-RL (part) improves Normal tasks substantially from 73.60 to 75.20, with multi-turn performance improved from 65.00 to 70.00, while maintaining 82.88 on Special subset, confirming that **targeted RL training enhances long-context reasoning and handling complex interaction, improving generalization on multi-turn tasks while maintaining stability on Special cases**. These results highlight the **advantage of combining SFT and RL** to improve function-calling abilities of the model.

Notably, both SFT and RL incur low training costs, as summarized in Appendix C.3.

6 Conclusion

This paper introduces GENESISFUNC, an automated data-generation pipeline for strengthening the function-calling abilities of LLMs. Beginning with a reliable tool set curated from open-source benchmarks, GENESISFUNC leverages a coordinated multi-agent framework that assists a dialogue generator in producing high-quality, diverse, and representative function-calling training data, while remaining model-agnostic in design. In extensive experiments, models trained with GENESISFUNC

achieve state-of-the-art performance, marking a concrete advance in tool-augmented AI agents, and the methodology can be extended to more agentic and complex function-calling tasks using proprietary or open-source LLMs in future work.

Limitations

Despite the effectiveness of GENESISFUNC, several important limitations still remain. First, GENESISFUNC-8B achieves competitive tool-use performance but still falls short of API-based models like GPT-4 in broader reasoning and comprehension. Enhancing general abilities alongside function-calling remains an open challenge. Second, our training data does not yet fully encompass highly complex multi-turn scenarios that require tightly coupled tool sequences. In future work, we plan to focus more on challenging settings, such as agentic workflows and complex benchmarks. Nevertheless, we firmly believe that, in principle, these more demanding scenarios can also be addressed by extending the methodology developed in this work.

Ethical Considerations

AI Assistant Use We used LLMs to assist with improving grammar, clarity, and wording in parts of this work. The use of LLMs was limited to language refinement, with all ideas, analyses, and conclusions solely developed by the authors.

References

Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, G. P. Shrivatsa Bhargav, Maxwell Crouse, R. Chulaka Gunasekara, Shajith Ikbal, Sachindra Joshi, Hima Karanam, Vineet Kumar, Asim Munawar, Sumit Neelam, Dinesh Raghu, Udit Sharma, Adriana Meza Soria, and 7 others. 2024. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: EMNLP 2024 - Industry Track, Miami, Florida, USA, November 12-16, 2024*, pages 1131–1139. Association for Computational Linguistics.

Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Vernon Austel, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis A. Lastras. 2024. API-BLEND: A comprehensive corpora for training and benchmarking API llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12859–12870. Association for Computational Linguistics.

André Bauer, Simon Trapp, Michael Stenger, Robert Leppich, Samuel Kounev, Mark Leznik, Kyle Chard, and Ian T. Foster. 2024. Comprehensive exploration of synthetic data generation: A survey. *CoRR*, abs/2401.02524.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. 2025a. Acebench: Who wins the match point in tool learning? *CoRR*, abs/2501.12851.

Hao Chen, Zhexin Hu, Jiajun Chai, Haocheng Yang, Hang He, Xiaohan Wang, Wei Lin, Luhang Wang, Guojun Yin, and Zhuofeng Zhao. 2025b. Toolforge: A data synthesis pipeline for multi-hop search without real-world apis. *CoRR*, abs/2512.16149.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit S. Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 81 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261.

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, Aston Zhang, and 82 others. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2024. Large language models can plan your travels rigorously with formal verification tools. *CoRR*, abs/2404.11891.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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. *CoRR*, abs/2308.00675.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

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. 2024. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 4363–4400. Association for Computational Linguistics.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516.

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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 3102–3116. Association for Computational Linguistics.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. Torl: Scaling tool-integrated RL. *CoRR*, abs/2503.23383.

Zongjie Li, Chaozheng Wang, Pingchuan Ma, Daoyuan Wu, Shuai Wang, Cuiyun Gao, and Yang Liu. 2024. Split and merge: Aligning position biases in llm-based evaluators. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 11084–11108. Association for Computational Linguistics.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. 2024. Hammer: Robust function-calling for on-device language models via function masking. *CoRR*, abs/2410.04587.

Zihan Lin, Xiaohan Wang, Hexiong Yang, Jiajun Chai, Jie Cao, Guojun Yin, Wei Lin, and Ran He. 2025. Awpo:enhancing tool-use of large language models through explicit integration of reasoning rewards. *CoRR*, abs/2512.19126.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2025a. Toolace: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Ziche Liu, Rui Ke, Yajiao Liu, Feng Jiang, and Haizhou Li. 2025b. Take the essence and discard the dross: A rethinking on data selection for fine-tuning 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, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 6595–6611. Association for Computational Linguistics.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh R. N., Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey. *Trans. Mach. Learn. Res.*, 2023.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and

Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *CoRR*, abs/2504.13958.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 Comput. Sci.*, 19(8):198343.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A graduate-level google-proof q&a benchmark. *CoRR*, abs/2311.12022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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. *CoRR*, abs/2306.05301.

Fanqi Wan, Weizhou Shen, Shengyi Liao, Yingcheng Shi, Chenliang Li, Ziyi Yang, Ji Zhang, Fei Huang, Jingren Zhou, and Ming Yan. 2025. Qwenlong-l1: Towards long-context large reasoning models with reinforcement learning. *CoRR*, abs/2505.17667.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. 2025. Logic-rl: Unleashing LLM reasoning with rule-based reinforcement learning. *CoRR*, abs/2502.14768.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40 others. 2025. Qwen3 technical report. *CoRR*, abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Dylan Zhang, Justin Wang, and François Charton. 2024. Instruction diversity drives generalization to unseen tasks. *CoRR*, abs/2402.10891.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaonkar, Rithesh R. N., Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, and 2 others. 2025. xlam: A family of large action models to empower AI agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 11583–11597. Association for Computational Linguistics.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 400–410. Association for Computational Linguistics.

Ruizhe Zhong, Xingbo Du, Shixiong Kai, Zhentao Tang, Siyuan Xu, Hui-Ling Zhen, Jianye Hao, Qiang Xu, Mingxuan Yuan, and Junchi Yan. 2024. LLM4EDA: emerging progress in large language models for electronic design automation. *CoRR*, abs/2401.12224.

A Details of GENESISFUNC

A.1 Multi-Agent Framework

System Prompts in Multi-Agent Framework In our proposed multi-agent framework, we employ four distinct agents, Sample Agent, Memory Agent, Function Agent, and Judge Agent, to assist dialogue generation by producing diverse and high-quality conversations. Figure 4 presents the complete system-level prompts that define the roles and behaviors of the Sample Agent, Memory Agent, Function Agent, and Judge Agent. These prompts are used as system prompts throughout the generation process, with bracketed segments serving as dynamically filled placeholders.

Details of Sample Agent To construct a sampled subset $\mathcal{T}_s \subseteq \mathcal{T}$ from the global **Tools Pool**, the sample agent employs GPT-4o (OpenAI, 2023) to evaluate the semantic and functional similarity between tools. Given two tool descriptions that specify their name, purpose, and parameter schema, the model assigns a similarity score $r \in [0, 1]$, where $r = 1$ indicates nearly identical functionality and $r = 0$ indicates entirely unrelated purposes. Each tool pair is scored with a fixed prompt and temperature $= 0$ to ensure deterministic outputs. Tools with $r > 0.75$ or sharing similar functions are grouped as target tools $\mathcal{T}_{\text{target}} = \{t_1, \ldots, t_m\}$. Distractors $\mathcal{T}_{\text{dist}} = \{t_{m+1}, \ldots, t_{m+n}\}$ are sampled from other clusters according to their relevance scores, categorized as high ($r > 0.6$), medium ($0.3 < r \leq 0.6$), and low ($r \leq 0.3$). All threshold values and sampling ratios were empirically tuned for consistency and reproducibility.

Details of the Judge Agent The Judge Agent selects the best dialogue from multiple candidates based on problem significance and tool appropriateness. Prior literature has documented that LLM-based evaluators can exhibit positional bias in pairwise comparisons, where judgments may systematically favor specific positions within prompts (Li et al., 2024). To mitigate this, candidate dialogues are randomly shuffled before being evaluated by the Judge Agent. Additionally, we perform an A/B swap test on a subset of pairs, where dialogues are evaluated under swapped ordering. We observe no consistent directional preference across swapped conditions, indicating that positional bias is unlikely to be a major confounding factor in our selection process. This analysis enhances the reliability of the Judge Agents decisions and helps ensure that the data pipelines improvements are not driven by spurious positional effects.

Reproducibility Our Multi-Agent Framework is model-agnostic and does not rely on any proprietary model capabilities or a specific combination of models. In practice, different stages of the pipeline impose different requirements on model behavior. In particular, tool-calling generation benefits from models with stable API-following performance, while other stages primarily require output consistency rather than advanced reasoning ability. In our implementation, Gemini-2.5 Pro (Comanici et al., 2025) is adopted for tool-calling related agents due to its stable behavior, which reflects an engineering choice rather than an algorithmic requirement. Moreover, replacing Gemini-2.5 Pro with open-source models such as Qwen3-32B (Yang et al., 2025) preserves the overall workflow and produces coherent training data, although data quality and downstream tool-calling performance are moderately reduced. Specifically, on BFCL, the performance decreases by about 0.5% on the non-live split and 0.9% on the live split. These results indicate that stronger models primarily affect data quality, while the pipeline logic and functionality remain unchanged, supporting both reproducibility and practical accessibility.

A.2 Dialogue Generation System

Algorithmic Workflow of Multi-Agent Dialogue Generation Algorithm 1 illustrates the pseudocode workflow of our multi-agent dialogue generation system, highlighting the interactions among different agents, the data flow between successive stages, and the overall generation and selection process.

Case Study We leverage a multi-agent framework to enable the dialogue generation system to produce three categories of dialogue: single-turn, multi-turn, and special cases. For both single-turn and multi-turn, we considered situations where users may accomplish either a single task or multiple tasks simultaneously. In addition, we incorporate special cases, such as when none of the available tools can address the user's request, or when the user's query prevents the model from filling in the tool parameters.

Figure 9 and Figure 10 illustrate single-turn scenarios in which the user intends to invoke one or multiple tools to complete either a single task

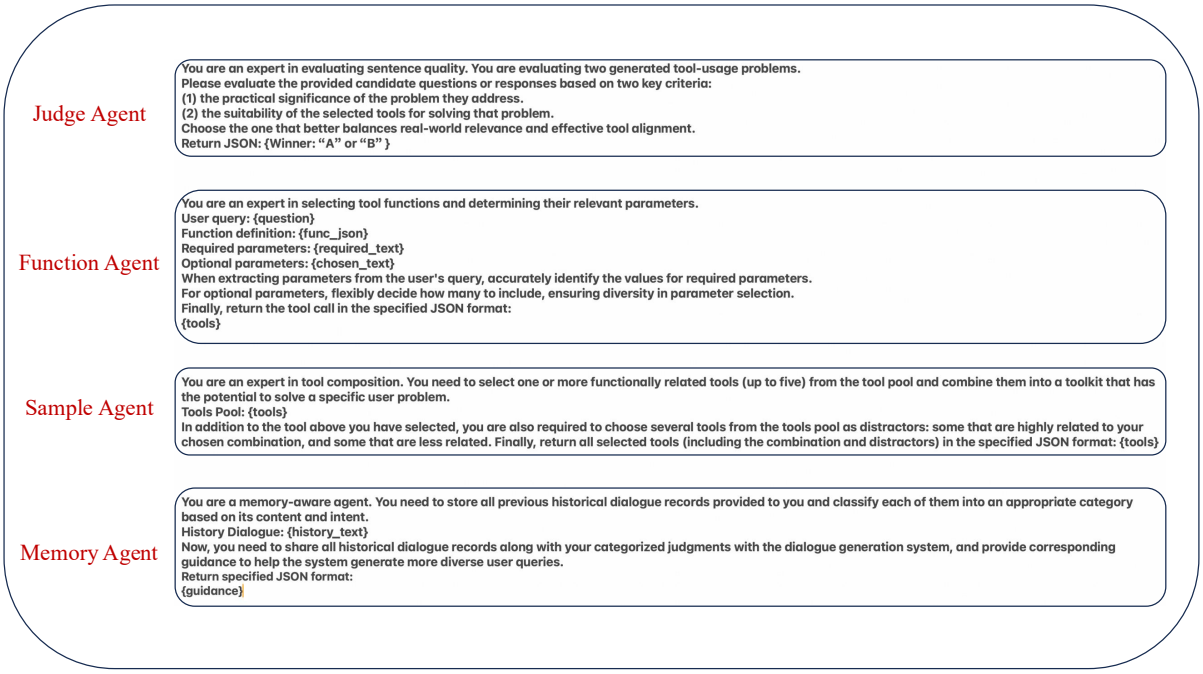


Figure 4: The system prompts of each agent in multi-agent framework.

or multiple tasks. Figure 11 and Figure 12 present analogous cases in multi-turn dialogues, where one or more tools are required to handle a single task or multiple tasks. Finally, Figure 13 demonstrates the special-case dialogue that we constructed.

A.3 Multi-Stage Evaluation

Rule Checker Table 5 outlines the check rules we use, which consist of four complementary aspects: tool definition completeness, call format and argument compliance, dialog structure soundness, and consistency between tool outputs and the assistants responses.

Model Checker The Model Checker verifies the correctness of function-calling dialogues using the following system-level prompt: *"You are a Model Checker responsible for verifying the correctness of a function-calling dialogue. Given the dialogue, tool specifications, and tool-call outputs, determine whether the assistant selected the appropriate tool, used correct parameter formats, and provided a faithful answer. Identify any semantic errors and assign a confidence score between 0 and 1."* This prompt defines the Model Checkers role and behavior across evaluations. We adopt GPT-4o (OpenAI, 2023) as the backend model due to its stable evaluative behavior, which improves robustness in automatic screening. This choice is an engineering consideration rather than an algorithmic requirement, and alternative models can be used without affecting the evaluation pipeline.

B Details of Training Data

Our constructed function-calling dataset consists of 2,000 single-turn dialogue scenarios, 2,000 multi-turn dialogue scenarios, and 500 special-case dialogues. To further demonstrate the strengths of our dataset in terms of quality, coverage, and diversity, we provide a set of statistical analyses.

Relationship Between Training Data and BFCL Evaluation Our training data uses the tool schemas provided by BFCL. During training, we do not use any BFCL test queries, test outputs, or instantiated evaluation samples. All dialogue are newly generated by our multi-agent framework, including user queries, dialogue contexts, and concrete parameter values. Although the tool schemas are shared between training and evaluation, this setting follows the standard in-domain evaluation protocol of BFCL and prior function-calling benchmarks. The BFCL evaluation focuses on generalization to unseen queries and novel argument combinations rather than memorization of specific tool invocations. As a result, the observed performance gains reflect improved tool selection and parameter grounding instead of exposure to evaluation instances.

Algorithm 1: Dialogue Generation with Multi-Agent Coordination

Input: Global tools pool $\mathcal{T}$; history $\mathcal{D}$ with type labels $\tau(\cdot) \in \mathcal{C}$; max turns $T_{\max}$; candidate count N

Output: Final dialogue `Dialog`; turn-level trajectory `traj`

$\mathcal{T}_s \leftarrow$ SAMPLEAGENT.SELECT($\mathcal{T}$) ; // targets + distractors

summary, forbidden $\leftarrow$ MEMORYAGENT.SUMMARIZE($\mathcal{D}$) ; // avoid seen types

Dialog $\leftarrow [\,]$; traj $\leftarrow [\,]$; $s_0 \leftarrow$ INITSTATE($\mathcal{T}_s$, summary)

for $t \leftarrow 0$ **to** $T_{\max}$ **do**

  $\mathcal{CAND} \leftarrow \varnothing$; // N candidates per round

  for $i \leftarrow 1$ **to** N **do**

    $q_t \leftarrow U$.ISSUE($\mathcal{T}_s, s_t$, summary, forbidden) ; // user request

    $a_t \leftarrow A$.PLAN(q_t, s_t) ; // $\mathcal{A}_{\text{asst}} = \{\text{ask}, \text{call}, \text{answer}\}$

    if $a_t =$ ask **then**

      $o_t \leftarrow U$.CLARIFY(a_t) ; // supply missing constraints

    else if $a_t =$ call **then**

      $\{t_1, \ldots, t_m\} \leftarrow$ FUNCTIONAGENT.SELECT($\mathcal{T}_s, q_t$) ; // one or more tools

      $\mathcal{P}' \leftarrow$ FUNCTIONAGENT.SLOTSELECT(q_t) ; // optional param subset

      args $\leftarrow$ FUNCTIONAGENT.FILL($q_t, \mathcal{A}'$) ; // instantiate required + optional

      $r_t \leftarrow$ FUNCTIONAGENT.EXEC($\{t_j\}$, args) ; // simulate

      $o_t \leftarrow A$.SUMMARIZE(r_t) ; // explicit tool outputs

    else

      $o_t \leftarrow A$.DIRECTANSWER(q_t) ; // non-tool reply

    cand$_i \leftarrow (q_t, a_t, o_t)$; $\mathcal{CAND} \leftarrow \mathcal{CAND} \cup \{\text{cand}_i\}$;

  cand* $\leftarrow$ JUDGEAGENT.SELECT($\mathcal{CAND}$; significance, suitability) ; // pick best of N

  Dialog $\leftarrow$ Dialog $\|$ cand*; traj $\leftarrow$ traj $\|$ cand* ;

  $\tau_{\text{new}} \leftarrow$ INFERTYPE(cand*); $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\text{cand}^*, \tau_{\text{new}})\}$;

  forbidden $\leftarrow$ forbidden $\cup \{\tau_{\text{new}}\}$; // diversity control

  $s_{t+1} \leftarrow$ UPDATESTATE(s_t, cand*) ; // record t, a, and payload a

  if STOP(s_{t+1}) **then**

    break

  ; // single-turn met / constraints satisfied / max turns

return Dialog, traj

Quality In terms of quality, our constructed dataset features an average of 3.11 tools used per dialogue, 4.46 turns per multi-turn scenario, and 3.27 tasks per multi-task case. This design enables each dialogue to cover multiple tools, resulting in a substantially larger number of tool invocation instances than the number of dialogues, and the high number of tool calls, longer dialogue lengths, and richer task compositions expose the model to more complex tool-using situations, thereby better stimulating and enhancing its function-calling capability, while ensuring all data are newly generated and do not overlap with evaluation samples.

Coverage In terms of coverage, our dataset incorporates a broader range of scenarios compared to previously constructed dataset. This expanded scenario coverage is a key factor in strengthening the model's function-calling ability. An overview of the data statistics used in these representative tool-augmented LLMs is presented in Table 6.

Diversity In terms of diversity, we further analyze the slot filling ratios to better demonstrate the diversity of our tool-call training data. Specifically, we sample 400 tools with concrete parameter values from the constructed dialogues. For each tool, we compute the slot filling ratio by dividing the number of filled optional parameters by the total number of available optional parameters, and then group the results into five equal-width intervals. Figure 5 reports the distribution across these ranges.

Table 5: Example rules for our Rule Checker in multi-stage evaluation.

Aspect	Rules
Tool Definition Completeness	Check that the tool schema parses and includes required fields (name, description, parameters). Verify that parameter specs declare type or format and required or optional flags. Ensure the tool name is unique within the tools pool.
Call Format & Parameter Compliance	Confirm the called tools exist in the tools pool and use the correct name. Ensure all required parameters appear exactly once and no unknown keys exist. Validate that values satisfy declared type/regex/range constraints (e.g., ISO date, enum).
Dialogue Structure Soundness	Check that role order is valid. Ensure no dangling tool calls and every tool output is referenced by the assistant. Verify turn length within configured limits.
Tool-Assistant Consistency	Ensure assistant summaries faithfully reflect tool outputs without invented fields or values. Verify error codes and messages are propagated correctly. Check that tool names match the actual call.

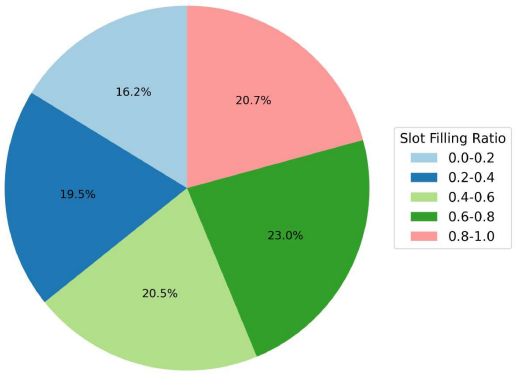


Figure 5: Distribution of slot filling ratios across 400 sampled tools.

The relatively balanced counts indicate that our dataset effectively captures heterogeneous slot utilization patterns, ranging from sparse filling to near-complete filling, highlighting its diversity.

Efficiency We examine the efficiency of the data construction process and compare generation cost with other synthetic datasets. Unlike synthetic function-calling datasets such as ToolLLM (Qin et al., 2024) and ToolACE (Liu et al., 2025a) that rely on manually designed or synthesized APIs, our approach operates directly on the tools already defined in BFCL (Patil et al., 2025). This design avoids additional tool construction and manual schema engineering, thereby reducing the cost of dataset creation. The generation cost of our approach arises from model generation within a multi-agent framework, while the tool space remains fixed and reusable across samples. Consequently, the overall construction cost scales linearly with the number of generated samples and does not introduce dataset-specific tool design or manual annotation overhead. Since existing datasets differ in tool complexity, agent design, and model backbones, a direct numerical comparison of generation cost is difficult. Nevertheless, under a unified benchmark setting, our approach achieves a balance between the quality of the training data and construction efficiency.

C Experimental Details

C.1 Benchmarks

BFCL The Berkeley Function-Calling Benchmark (BFCL) (Patil et al., 2025) is a comprehensive framework for evaluating the function-calling abilities of LLMs across multiple languages, domains, and complex scenarios. It includes 4,951 test cases, with 3,951 single-turn dialogs and 1,000 multi-

Table 6: Comparison of dialogue scenario coverage across representative tool-augmented datasets. Here, *Single-Single* denotes single-turn dialogues designed to accomplish a single task. *Single-Multi* refers to single-turn dialogues that involve multiple tasks within the same interaction. *Multi-Single* indicates multi-turn dialogues focused on a single task. *Multi-Multi* represents multi-turn dialogues that cover multiple tasks. and *Special-Case* corresponds to challenging scenarios such as the absence of applicable tools or cases where parameter filling is infeasible. Our dataset uniquely covers all five categories.

Dataset	Single-Single	Single-Multi	Multi-Single	Multi-Multi	Special-Case
Gorilla (Patil et al., 2024)	✓	×	×	×	×
ToolAlpaca (Tang et al., 2023)	✓	✓	×	×	×
ToolLLM (Qin et al., 2024)	✓	✓	×	×	×
xLAM (Liu et al., 2024)	✓	✓	✓	×	×
ToolACE (Liu et al., 2025a)	✓	✓	✓	×	✓
GENESISFUNC	✓	✓	✓	✓	✓

turn dialogs that emphasize dynamic and realistic settings. The tools pool is broad, combining **Non-Live** tools curated to cover diverse situations with **Live** tools that are continuously uploaded by users. BFCL supports two evaluation methods, one based on Python and the other not. In this work, we adopt the Python-based approach, which is shown below:

- **Simple Function:** This category represents the most basic yet also the most frequently encountered setting, where the input explicitly contains exactly one json function description and the model is expected to correctly invoke precisely that single function.
- **Multiple Function:** In this setting, the model receives 2 to 4 json function descriptions, but only one of them should be invoked. The task requires the model to identify and select the most suitable function call based on the users query and context.
- **Parallel Function:** Here, the model must invoke multiple function calls simultaneously in response to a single user query. The challenge lies in determining how many calls are required and executing them in parallel, whether the query is phrased as a short request or a longer description.
- **Parallel Multiple Function:** This combines the complexity of both multiple and parallel function settings. The model is given several function descriptions and must decide, for each one, whether it should be invoked, possibly multiple times or not at all.

For every category, BFCL provides both AST-based evaluation and a corresponding executable evaluation. In the executable setting, Python functions are manually implemented, inspired by publicly available REST API endpoints (such as retrieving weather data) as well as directly computable functions (such as linear regression). The purpose of this executable track is to assess whether the generated function calls can be reliably applied in real-world applications that depend on stable function execution.

API-Bank The API-Bank (Li et al., 2023) is composed of 314 dialogues involving a total of 753 API calls, specifically constructed to evaluate the abilities of LLMs in planning, retrieving, and invoking APIs in diverse scenarios. Within the dataset, 363 cases require only a single call, while 122 instances involve multiple calls, reflecting both simple and more complex usage patterns. Performance is systematically measured along three distinct dimensions, providing a comprehensive assessment of tool-use capability:

- **Call:** Evaluates whether the language model can correctly invoke a known API based on a given query.
- **Retrieval+Call:** Evaluates the model's ability to first identify and accurately retrieve the correct API from context and then successfully perform the corresponding call when the API is not provided.
- **Plan+Retrieval+Call:** Assesses the capacity to plan a sequence of actions, retrieve multiple APIs, and invoke them when the APIs are initially unknown.

The primary evaluation metric for API-Bank is *accuracy*, formally defined as the ratio of

Table 7: Configuration of hyper-parameters for model training.

Learning Rate	Warmup Ratio	LR Scheduler	Batch Size	Epochs	LoRA Rank	LoRA Alpha
10^{-4}	0.1	cosine	48	3	16	32

correctly generated predictions to the total number of evaluation attempts.

ACEBench ACEBench (Chen et al., 2025a) is a bilingual Chinese and English benchmark for assessing LLMs tool use ability under realistic conditions. It contains about 2,000 annotated instances spanning a broad API set and organizes evaluation into three tracks:

- **Normal:** Single/multi-turn cases with similar-API and preference settings; calls are checked via AST matching against gold annotations.
- **Special:** Inputs with missing, malformed, or otherwise irrelevant parameters are included to thoroughly test the models robustness when handling imperfect or noisy instructions.
- **Agent:** Multi-turn, multi-step interactions in sandboxed scenarios, measuring process correctness and end-to-end task success.

Overall performance is computed from track-wise accuracies, providing fine-grained diagnostics of tool-use failures while avoiding reliance on live APIs or external LLM graders.

C.2 Baselines

Here we mainly introduce the two open-source models fine-tuned on function-calling data that we use as baselines, while details of the other open-source models and API-based models can be found in their publicly available technical reports.

- **ToolACE-8B** (Liu et al., 2025a): Obtained by fine-tuning LLaMA3.1-8B-Instruct with function-calling training data generated through data pipeline ToolACE.
- **Qwen-ToolRL:** Obtained by fine-tuning Qwen3-8B with the publicly available function-calling training dataset ToolRL (Qian et al., 2025), which is a hybrid corpus sampling 2K examples from ToolACE and 1K each from Hammer (Lin et al., 2024) and xLAM (Zhang et al., 2025).

C.3 Compute Cost

We apply the parameter-efficient LoRA (Hu et al., 2022) strategy for fine-tuning and perform SFT using the LLaMA-Factory framework (Zheng et al., 2024). The primary computational overhead during generation arises from API calls, while post-training costs remain lightweight. Specifically, SFT is completed on a single A800 GPU in about 30 minutes, and the RL stage on four A800 GPUs in roughly 4 hours, making the overall compute requirements practical for reproduction. The training data follows the Alpaca format, where the instruction includes the system prompt, tool pool, and user query, and the output contains the tool call. Hyper-parameters are shown in Table 7.

C.4 Statistical Significance and Variance Analysis

All results reported in the main text are averaged over three runs with different random seeds. We also compute the standard deviation (reported as $\pm$SD in the tables) to assess model stability. Across all benchmarks, GENESISFUNC-8B shows small standard deviations, typically below 0.5, indicating consistent performance across runs. The observed performance gains of GENESISFUNC-8B exceed twice the standard deviations on all benchmarks, demonstrating that the improvements are well beyond random variation. To further verify robustness, we perform paired t-tests between GENESISFUNC-8B and the strongest open-source baseline (ToolACE-8B), showing statistically significant gains ($p < 0.05$).

D Additional Experiments

D.1 Study on General Abilities

To assess the broader impact of GENESISFUNC-8B on general abilities, we evaluated it on five benchmarks: MMLU for knowledge (Hendrycks et al., 2021), EvalPlus for code generation (Liu et al., 2023), GSM8K for mathematics (Cobbe et al., 2021), MGSM for multilingual reasoning (Shi et al., 2023), and GPQA for logical reasoning (Rein et al., 2023). Baselines included LLaMA-3.1-8B-Instruct (Dubey et al., 2024), Qwen3-8B (Yang et al., 2025), ToolACE-8B (Liu et al., 2025a),

Table 8: Ablation study of different slot selection strategies in the function agent. To compare with our proposed dynamic slot selection approach, we also adopt two fixed strategies: filling all optional parameters and leaving them empty, and evaluate on BFCL. The best results are marked in bold, and the second best are underlined.

Models	Non-Live					Live				
	Simple	*Multiple*	*Parallel*	*Parallel Multiple*	*Overall*	*Simple*	*Multiple*	*Parallel*	*Parallel Multiple*	*Overall*
Qwen3-8B	78.92	95.00	91.50	89.00	88.60	80.23	77.21	81.25	75.00	77.79
ToolACE-8B	81.17	96.00	94.00	**93.00**	91.04	82.95	79.58	75.25	**85.12**	80.73
GENESISFUNC-8B(dynamic)	**86.85**	**97.25**	**96.15**	**93.00**	**93.31**	**88.25**	**80.50**	**81.50**	84.88	**83.78**
GENESISFUNC-8B(max)	86.00	96.85	95.50	92.25	92.65	86.80	80.00	81.25	83.86	82.98
GENESISFUNC-8B(min)	85.75	97.00	95.75	92.00	92.63	87.10	79.50	81.00	84.07	82.92

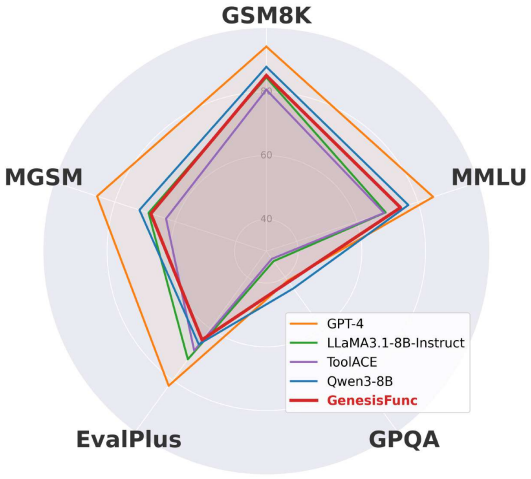


Figure 6: General abilities of models. Evaluation are conducted in five dimensions.

and GPT-4 (OpenAI, 2023). As shown in Figure 6, GENESISFUNC-8B performs on par with Qwen3-8B on most benchmarks, indicating that our fine-tuning substantially improves function-calling ability while preserving the models general abilities. It also surpasses similarly sized open-source models on many tasks, demonstrating competitive performance at the 8B scale. The remaining gap to GPT-4 in reasoning and comprehension is expected and likely stems from differences in model size and the breadth of training data rather than negative transfer from function-calling training. Overall, these results highlight the promise of targeted specialization for function-calling while maintaining broad competence, and they leave open the challenge of jointly improving multiple capabilities together with function-calling performance in a single model.

D.2 Different Slot Selection Strategies in the Function Agent

Table 8 presents the results of different slot selection strategies in the function agent. Compared with two fixed strategies, filling all optional parameters (max) and leaving all optional parameters empty (min), our proposed dynamic slot selection approach achieves the best overall performance on both the Non-Live and Live settings. Specifically, GENESISFUNC-8B(dynamic) reaches an overall accuracy of 93.31 on the Non-Live split and 84.83 on the Live split, surpassing both fixed strategies. These results demonstrate that dynamically selecting relevant slots enables the model to better capture diverse tool-use patterns while avoiding unnecessary noise, thereby improving the robustness and generalization of function-calling.

D.3 Ablation on Multi-Stage Evaluation Module

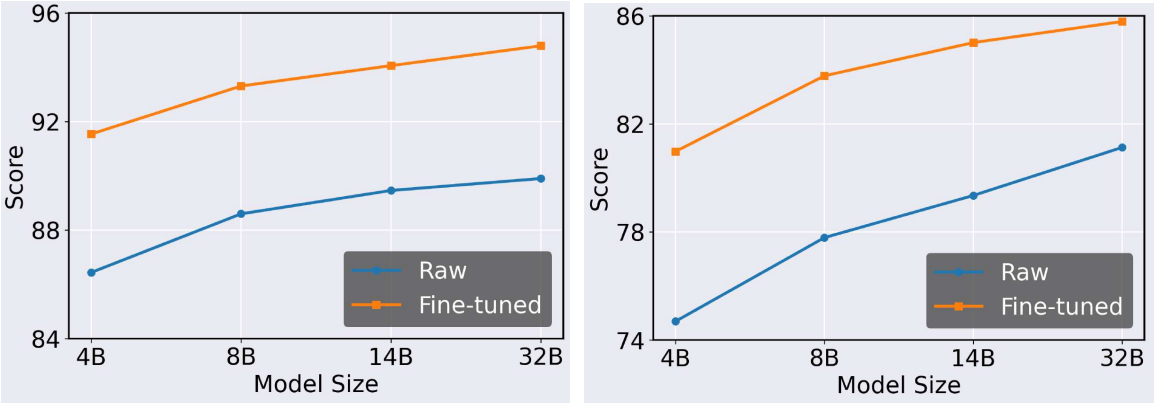
As discussed earlier, we employ a multi-stage evaluation module to ensure the correctness of the dialogues generated in the previous stage. To evaluate its effectiveness, we fine-tune models on two datasets: one that has passed through this evaluation module and another that has not. The fine-tuned models are then evaluated on the BFCL benchmark, with results presented in Table 9. The comparison clearly shows that models trained on data evaluated by the multi-stage module achieve higher overall accuracy than those trained on non-evaluated data, thereby demonstrating the effectiveness of our proposed evaluation module.

D.4 Scaling to Different Model Sizes

Scaling laws suggest a close relationship between model capacity and empirical performance. To examine how function-calling ability scales with model size, we evaluate the Qwen-3-xB model family (Yang et al., 2025), covering a range of parameter scales (4B, 8B, 14B, and 32B). Both raw models and models fine-tuned on our dataset

Table 9: Ablation study of multi-stage evaluation module. We remove the multi-stage evaluation module, and evaluate on BFCL. The best results in each category are marked in bold. The second best results are underlined.

Models	Non-Live					Live				
	Simple	*Multiple*	*Parallel*	*Parallel Multiple*	*Overall*	*Simple*	*Multiple*	*Parallel*	*Parallel Multiple*	*Overall*
Qwen3-8B	78.92	95.00	91.50	89.00	88.60	80.23	77.21	81.25	75.00	77.79
ToolACE-8B	81.17	96.00	94.00	**93.00**	91.04	82.95	79.58	75.25	**85.12**	80.73
GENESISFUNC-8B	**86.85**	**97.25**	**96.15**	**93.00**	**93.31**	**88.25**	**80.50**	**81.50**	84.88	**83.78**
- multi-stage evaluation module	85.17	97.00	96.00	92.50	92.68	87.00	80.00	81.25	84.05	83.08



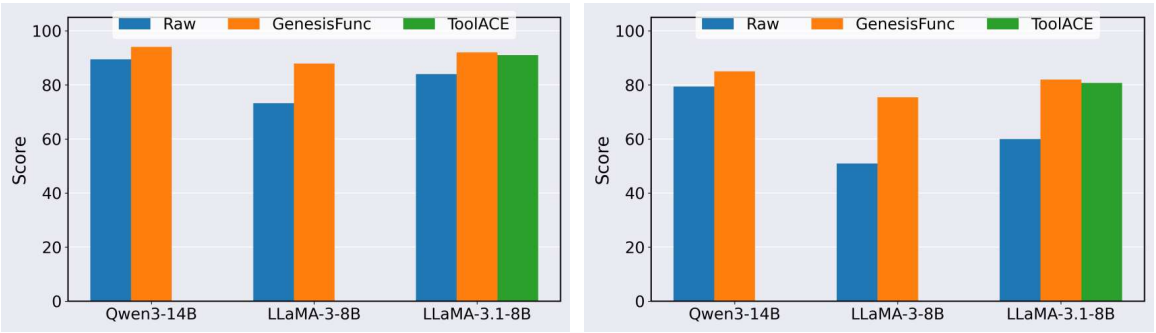
(a) Non-Live (b) Live

Figure 7: Scaling analysis of function-calling performance. We evaluate raw and fine-tuned Qwen-3-xB models of different sizes on the BFCL benchmark in (a) Non-Live and (b) Live settings.

are assessed on the BFCL (Patil et al., 2025), and the corresponding results are presented in Figure 7. Overall, the results show a steady improvement in function-calling performance as model size increases across both non-live and live evaluation settings, with larger models exhibiting stronger robustness and more consistent execution accuracy. In comparison to raw models, fine-tuned models consistently achieve higher performance at all scales, indicating that supervision from the GENESISFUNC effectively enhances function-calling behavior. Importantly, the fine-tuned models maintain a smooth and stable scaling pattern, suggesting that GENESISFUNC complements model capacity rather than altering the underlying scaling dynamics. Together, these observations demonstrate the effectiveness of GENESISFUNC in supporting scalable performance gains for LLMs.

D.5 Generalization across Model Backbones

To analyze the impact of backbone choice on function-calling performance, we evaluate a set of representative LLMs, including Qwen3-14B, LLaMA-3-8B-Instruct, and LLaMA-3.1-8B-Instruct. All models are fine-tuned using our dataset and evaluated on the BFCL benchmark. In addition, to mitigate potential confounding effects introduced by different backbones and to more rigorously isolate the contribution of the data-generation pipeline itself, we conduct a controlled comparison on the same backbone. Specifically, since ToolACE is trained on LLaMA-3.1-8B-Instruct, we directly compare the two data-generation pipelines on this backbone. The comparative results are summarized in Figure 8. Figure 8 shows that fine-tuning with GENESISFUNC consistently improves performance across all examined backbones under both non-live and live evaluation settings, demonstrating strong cross-backbone robustness. Despite differences in model architectures and pre-training objectives, fine-tuning yields stable and significant gains overall. Notably, models with lower initial function-calling accuracy tend to exhibit larger relative improvements, thereby narrowing the performance gap across backbones. More importantly, the controlled comparison on LLaMA-3.1-8B-Instruct indicates that, under the same backbone, GENESISFUNC outperforms ToolACE, providing stronger evidence that the observed



(a) Non-Live (b) Live

Figure 8: Generalization across model backbones. We evaluate raw and fine-tuned LLMs with different backbone architectures on the BFCL benchmark in (a) Non-Live and (b) Live settings.

gains stem from the data-generation pipeline rather than backbone-specific characteristics.

E Details of Reinforcement Learning

E.1 GRPO Algorithm

To fine-tune the model with structured rewards, we adopt *Grouped Relative Policy Optimization* (GRPO), a variant of PPO that normalizes advantages within groups of responses derived from the same input query. This design mitigates variance across samples under identical contexts, thereby stabilizing and accelerating training. Specifically, for each query Q, the rollout responses form a group $G_Q = \{(s_1, r_1), (s_2, r_2), \ldots, (s_n, r_n)\}$, where s_i denotes a candidate response and r_i its reward. Each reward is obtained as the sum of correctness and formatting scores relative to the reference annotation. Within each group, the mean μ_Q and standard deviation σ_Q of rewards are computed as:

$$\mu_Q = \frac{1}{n}\sum_{i=1}^{n} r_i, \quad \sigma_Q = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \mu_Q)^2}, \tag{2}$$

and the normalized advantage of response s_i is defined as:

$$A(s_i|Q) = \frac{r_i - \mu_Q}{\sigma_Q + \eta}, \tag{3}$$

where η is a small constant added for numerical stability.

The policy π_θ is then updated using the clipped PPO objective with group-normalized advantages:

$$r_\theta(s_i|Q) = \frac{\pi_\theta(s_i|Q)}{\pi_{\text{old}}(s_i|Q)}. \tag{4}$$

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\big[\min(r_\theta(s_i|Q)\,A(s_i|Q), \text{clip}\left(r_\theta(s_i|Q),\,1-\epsilon,\,1+\epsilon\right) \times A(s_i|Q))\big]. \tag{5}$$

Unlike standard PPO, GRPO omits the KL penalty against a reference model, thereby allowing greater flexibility in adapting to diverse custom reward functions while still retaining training stability. This modification improves overall sample efficiency, encourages stronger alignment with structured reward signals, and leads to faster convergence as well as more robust policy performance.

E.2 Reward Design

Reward functions play a central role in reinforcement learning, guiding models to generate outputs that are both valid and useful. Following prior studies on rule-based reward signals (Xie et al., 2025; Jin et al., 2025; Li et al., 2025; Qian et al., 2025), we adopt a two-dimensional design that combines structural compliance with functional correctness, tailored to the demands of tool-augmented dialogue.

Structural Compliance To ensure that generated outputs conform to the expected schema, we introduce a *structural compliance reward*. This reward evaluates whether each predicted tool call follows the prescribed format, including the presence of all mandatory fields and the correct logical ordering of elements. For single-tool cases, the reward reduces to a simple binary check defined as

$$r^{(i)}_{\text{structural}} = \begin{cases} 1, & \text{if the } i\text{-th output follows} \\ & \text{the required schema,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

```
{
  "question": "I'm looking for the highest-rated Thai restaurant in Austin that has
  outdoor seating for dinner tonight.",
  "function_calls": [
    {
      "function": "get_restaurant",
      "description": "Retrieve the highest-rated Thai restaurant in Austin that
      offers outdoor seating.",
      "parameters": {
        "cuisine": "Thai",
        "location": "Austin",
        "condition": "outdoor seating"
      }
    }
  ]
}
```


Figure 9: Example of a single-task scenario within single-turn dialogue.

For multi-tool cases, the score is computed for each tool call individually according to the above rule and then averaged across all calls:

$$R_{\text{structural}} = \frac{1}{N} \sum_{i=1}^{N} r_{\text{structural}}^{(i)}, \tag{7}$$

where N denotes the number of tools in the output.

Functional Correctness In addition to structural validity, we further assess whether the predicted tool calls can successfully achieve the intended functionality. For this purpose, we introduce a *functional correctness reward*. This reward explicitly captures the degree of semantic and functional agreement between the predicted calls and the reference calls. For a single-tool case, the reward is defined as follows:

$$r_{\text{correct}}^{(i)} = \begin{cases} 3, & \text{if the tool name and all parameters match exactly,} \\ 2, & \text{if the tool name is correct and at least one parameter matches,} \\ 1, & \text{if only the tool name is correct,} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

For multi-tool cases, the correctness score is first computed for each tool call using the above rule, and then the results are averaged across all calls:

$$R_{\text{correct}} = \frac{1}{N} \sum_{i=1}^{N} r_{\text{correct}}^{(i)}, \tag{9}$$

where N denotes the number of tools in the output.

Finally, we combine the two components to form the overall reward function. The structural compliance reward ensures that outputs remain syntactically valid, while the functional correctness reward encourages accurate execution of tool calls. By integrating both signals, the model receives feedback that jointly emphasizes format validity and semantic accuracy. The final reward is simply defined as follows:

$$R_{\text{final}} = R_{\text{structural}} + R_{\text{correct}}. \tag{10}$$

This unified formulation provides a balanced training signal, preventing the model from generating structurally invalid outputs and at the same time guiding it toward functionally reliable tool usage.

```
{
  "question": "I'm a historical fiction writer creating a new world and need
  detailed information on two major conflicts to use as inspiration. First, I want
  to understand the Hundred Years' War. Can you give me a general overview of it? I
  also need a profile on one of its key leaders, Edward the Black Prince, and a
  detailed account of the Battle of Agincourt. Second, for a different era, I need
  a summary of the Second Punic War. Please also provide a biography of Hannibal
  Barca, focusing on his military genius, and a tactical breakdown of the Battle of
  Cannae.",
  "function_calls": [
    {
      "function": "european_history.war_details",
      "description": "Get a general overview of the Hundred Years' War for a
      historical fiction writer.",
      "parameters": {
        "war": "Hundred Years' War"
      }
    },
    {
      "function": "european_history.leader_info",
      "description": "Get a profile on Edward the Black Prince, a key leader in the
      Hundred Years' War.",
      "parameters": {
        "leader": "Edward the Black Prince"
      }
    },
    {
      "function": "european_history.battle_details",
      "description": "Get a detailed account of the Battle of Agincourt.",
      "parameters": {
        "battle": "Battle of Agincourt"
      }
    }
  ]
}
```


Figure 10: Example of a multi-task scenario within single-turn dialogue.


```
{
  "question": "user: I'm planning a small garden and need to calculate the area of
  a triangular section.\nsystem: I can certainly help with that. What is the base
  measurement of the triangle?\nuser: The base is 15 meters long.\nsystem:
  Understood, a base of 15 meters. What is the height of the triangle?\nuser: The
  height is 8 meters. Please calculate the area for a triangle with base=15,
  height=8, and unit='meters'.",
  "function_calls": [
    {
      "function": "calculate_triangle_area",
      "description": "Calculate the area of a triangle given its base and height.",
      "parameters": {
        "base": 15,
        "height": 8,
        "unit": "meters"
      }
    }
  ]
}
```


Figure 11: Example of a single-task scenario within multi-turn dialogue.

```
{
  "question": "user: Hi, I'm thinking about buying an investment property and need
  to run some numbers. Can you help me figure out what my monthly loan payments
  would be?\nsystem: Of course. To calculate the monthly loan repayment, I'll need
  the total loan amount, the annual interest rate, and the term of the loan in
  years.\nuser: The loan amount is $400,000 with a 3.5% interest rate over 30
  years. While you're at it, can you also tell me how the property's value might
  change over that same time period?\nsystem: I can calculate the property's
  depreciated value. For that, I need the initial cost of the property and the
  annual depreciation rate. I can also adjust the initial cost for inflation over
  that period if you provide an estimated annual inflation rate.\nuser: Great. The
  property's initial cost is $500,000, and let's use a 2% annual depreciation rate.
  First, calculate the annual depreciation over 30 years. Then, do a second
  calculation for the monthly depreciation over the same 30 years. Also, adjust the
  initial sum of $500,000 for inflation over 30 years using a 2.5% inflation rate.
  Finally, please confirm the monthly loan repayment for a $400,000 loan, with a
  30-year loan term and an interest rate of 3.5%.",
  "function_calls": [
    {
      "function": "finance.property_depreciation",
      "description": "Calculates the annual depreciated value of a $500,000
      property over 30 years with a 2% annual depreciation rate.",
      "parameters": {
        "initial_cost": 500000,
        "depreciation_rate": 2,
        "years": 30
      }
    },
    {
      "function": "finance.inflation_adjustment",
      "description": "Adjusts an initial sum of $500,000 for inflation over 30
      years at a 2.5% annual rate.",
      "parameters": {
        "initial_sum": 500000.0,
        "years": 30
      }
    }
  ]
}
```


Figure 12: Example of a multi-task scenario within multi-turn dialogue.


```
{
  "question": "I'm planning a trip to Paris and want to find the best museum. Can
  you find one that's particularly famous for its reputation?",
  "function_calls": [
    {
      "function": "get_restaurant",
      "description": "Retrieve highest rated restaurant given cuisine, location,
      and a condition.",
      "parameters": {
        "cuisine": "",
        "location": "",
        "condition": ""
      }
    }
  ]
}
```


Figure 13: Example of a special-case dialogue.