TOOLMIND: A Large-Scale, High-Quality Tool-Use Dataset with Multi-Agent Simulation and Turn-Level Filtering

Anonymous ACL submission

Abstract

Large Language Model (LLM) agents have developed rapidly in recent years to solve complex real-world problems using external tools. However, the scarcity of high-quality trajectories still hinders the development of stronger LLM agents. To tackle this problem, we introduce ToolMind, a large-scale, high-quality tool-agentic dataset comprising 360k instances, including 160k synthetic data generated using over 14k tools and 200k augmented instances derived from open-source data. Our data synthesis pipeline first constructs a function graph based on parameter correlations and then uses a multi-agent framework to simulate realistic user–assistant–tool interactions. Beyond trajectory-level validation, we employ fine-grained turn-level filtering to remove erroneous or suboptimal steps, ensuring that only high-quality reasoning traces are retained. This approach mitigates error amplification during training while preserving self-corrective reasoning signals essential for robust tool-use learning. Models fine-tuned on ToolMind show significant improvements over baselines on τ-bench, τ^2-bench, and BFCL-v4. The dataset is provided at: https://anonymous.4open.science/r/ToolMind-sample-CDF3.

1 Introduction

Large Language Models (LLMs) have adopted Function Calling (FC) paradigm to interact with external environments, significantly expanding their capacity to solve complex real-world tasks (Zhao et al., 2023; Qu et al., 2025; Guo et al., 2025b; Qin et al., 2024; Ferrag et al., 2025). This capability hinges on the model's proficiency in tool use, which requires reliable interpretation of user instructions, appropriate function selection, and precise parameter generation. A data-centric approach has proven effective in enhancing such capabilities, typically requiring large-scale, high-quality datasets with diverse and accurate function-call demonstrations.

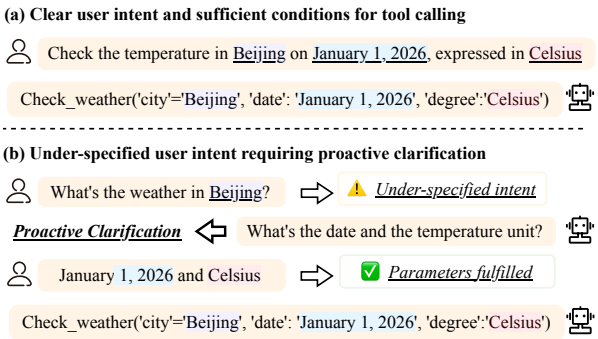

Figure 1: Comparison between (a) Direct Execution and (b) Proactive Clarification for under-specified requests.

Recent studies have actively synthesized data for various function calling scenarios (Liu et al., 2024b; Xi et al., 2024; Qin et al., 2023; Li et al., 2023a), spanning diverse application domains and interaction modalities such as selection from large toolsets, parallel function calls, and rejection of unspecified requests (Liu et al.; Prabhakar et al., 2025; Xu et al., 2025b). However, as summarized in Table 1, existing datasets have several critical limitations: limited scale, the absence of explicit reasoning traces, a lack of multi-turn interactions, and insufficient query diversity. These issues highlight the importance of high-quality, large-scale datasets to develop robust and generalizable tool-use models and agents.

A key challenge in constructing such datasets lies in capturing the dynamic nature of real-world user–assistant interactions (Huang et al., 2024). In practice, user requests are often incomplete or ambiguous (e.g., a query like "What's the weather in Beijing?" does not specify a time frame), requiring the model to actively elicit missing information and refine its understanding through multi-turn dialogue. However, most existing synthetic datasets simplify this process into single-turn or direct execution settings, thereby overlooking critical interaction patterns necessary for robust tool use. As

Dataset	# Sample	# Multi Turn	# Tools	Turn-Level Judge	User Engagement	Rejection / Clarification	CoT
ToolACE (Liu et al., 2024b)	11,300	794	16,388	x	x	x	x
When2Call (sft) (Ross et al., 2025)	27,952	0	3,803	x	x	x	x
xlam-function-calling-60k (Liu et al.)	60,000	0	3,605	x	x	x	x
APIGen-MT-5K (Prabhakar et al., 2025)	5,000	4,919	107	x	✓	✓	x
BUTTONInstruct (Chen et al., 2025b)	8,000	6,555	13,553	x	x	x	✓
TOUCAN (sft) (Xu et al., 2025b)	119,287	45,929	2,303	x	✓	✓	x
ToolMind-Synthetic (This work)	163,180	128,714	14,733	✓	✓	✓	✓

Table 1: Comparison between ToolMind and representative open-source tool-agentic datasets. "Turn-level Judge" denotes whether turn-level correctness annotations are available; "User Engagement" denotes whether users actively participate in multi-turn interactions; and "Rejection/Clarification" denotes whether the assistant performs clarification or follow-up questioning when handling unresolved requests.

illustrated in Figure 1, it's important to explicitly addresses this gap by incorporating trajectories that feature both direct execution and reasoning-driven clarification processes.

In this work, we introduce ToolMind, a large-scale, high-quality tool-use dataset with 360k samples. ToolMind comprises 160k synthesized interaction trajectories generated via multi-agent simulation, along with 200k augmented instances derived from open-source data. To ensure diversity and compositionality, we collect over 14k functions and construct a function graph that captures correlations among tool parameters, enabling the generation of complex function chains through graph-based sampling.

Building on this foundation, we develop a multi-agent simulation framework that models three distinct roles—user, assistant, and tool—each instantiated by an LLM to emulate realistic interaction dynamics. This framework enables the generation of trajectories that include both direct execution and clarification-driven reasoning processes. Furthermore, we apply a rigorous two-stage filtering pipeline, consisting of trajectory-level validation and fine-grained turn-level quality control, to eliminate erroneous or suboptimal steps while preserving high-quality reasoning signals.

Our experiments demonstrate the effectiveness of the ToolMind dataset in enhancing the tool-use capabilities of LLMs. With supervised fine-tuning (SFT), models of various scales show significant improvements over baselines across multiple benchmarks, including BFCL-v4 (Patil et al.), τ-bench (Yao et al., 2024), and τ^2-bench (Barres et al., 2025). Our main contributions are summarized as follows:

- **High-Quality Tool-Use Dataset.** We present ToolMind, a 360k-sample dataset that systematically addresses key limitations of existing resources, including insufficient scale, lack of reasoning traces, limited multi-turn interactions, and low query diversity.

- **Data Synthesis and Quality Filtering Pipeline.** We propose a novel pipeline that combines function graph–based sampling with multi-agent simulation, enabling the generation of diverse and realistic tool-use trajectories. We also introduce a two-stage filtering process that ensures high data quality while preserving essential reasoning and interaction signals.

- **Effectiveness of Large-Scale Simulation.** We show that ToolMind delivers significant and consistent performance improvements across models of various scales. These results empirically validate that large-scale synthetic data generated from simulated environments can serve as an effective alternative to real-world data for achieving substantial gains in model performance.

2 Related Works

LLM function calling. The integration of external tools and functions into language model workflows marks a major step toward extending model capabilities beyond text generation. This capability allows models to interact with real-world external environments, thereby enabling them to solve more complex, grounded, and action-oriented tasks (Huang et al., 2025; Feng et al., 2025b; Jin et al., 2025). Early work focused on enabling models to understand tool specifications, generate accurate function-call arguments, and obtain observations from external tools in the ReAct (Yao et al., 2022) style (Schick et al., 2023; Li et al., 2023b; Tang et al., 2023). More recently, with the rise

of reasoning-oriented models, function calling has been integrated as a native capability in chat templates and inference pipelines (Qian et al., 2025a,b; Wei et al., 2025; Yang et al., 2025b,a; Team et al., 2025; Zeng et al., 2025a). Through interleaved reasoning and acting, models alternate between reasoning and function calls, which enables them to handle long-horizon and complex tasks more effectively. Many studies inject function calling capabilities into models by automatically constructing specific tool-use datasets (Chakraborty et al., 2025; Feng et al., 2025a; Fang et al., 2026). However, the limited diversity and realism of training data remain major bottlenecks in further improving function calling performance.

Data synthesis for function calling. Current data synthesis approaches for function calling typically start from a predefined set of tools or APIs, followed by selecting relevant tools and synthesizing executable interaction trajectories (Chen et al., 2025a; Zeng et al., 2025b; Wang et al., 2025; Xu et al., 2025a; Lin et al., 2024). For instance, APIGen (Liu et al.) and ToolACE (Liu et al., 2024b) synthesize function-calling data through agentic pipelines that generate tool-use interactions grounded in executable APIs and validated calls. Later studies, such as BUTTONInstruct (Chen et al., 2025b) and APIGen-MT (Prabhakar et al., 2025), further extend this paradigm to compositional and multi-turn settings through structured task construction. However, existing function-calling datasets still have several limitations: limited scale and diversity, insufficient multi-turn interactions, missing reasoning traces, and overly explicit user intents that fail to reflect real-world ambiguity. In contrast, ToolMind addresses these limitations by emphasizing realistic multi-turn interactions, explicit reasoning traces, and fine-grained turn-level quality control, rather than relying solely on trajectory-level synthesis and validation.

3 Methodology

In this section, we present our data synthesis pipeline, including graph construction and function-chain sampling, multi-agent, multi-turn trajectory synthesis, and two-stage quality filtering. The overall synthesis pipeline is illustrated in Figure 2. All prompts used for trajectory synthesis and quality filtering are provided in Appendix B.

3.1 Graph Construction and Function Chain Sampling

We construct a directed graph over the collected functions to model their input–output compatibility, and then sample function chains via random walks for trajectory generation.

3.1.1 Function Collection and Preprocessing

Existing open-source work has already achieved substantial tool diversity. Building upon these resources, we collect functions from public datasets—including *xlam-function-calling-60k*[*], *glaive-function-calling-v2*[†], and *ToolACE*[‡]. The resulting function set covers a broad range of function types, complexity levels, and application domains, spanning everyday utilities to domain-specific APIs. We denote the collected function set by $\mathcal{F} = \{f_i\}_{i=1}^{N}$, where each function f_i has an input space $\mathcal{X}_i$ and an output space $\mathcal{Y}_i$, and may take multiple arguments, written as:

$$f_i : \mathcal{X}_i \to \mathcal{Y}_i, \qquad \mathcal{X}_i = \prod_{k=1}^{m_i} x_{i,k}, \;\; \mathcal{Y}_i = \prod_{\ell=1}^{n_i} y_{i,\ell}.$$

To make this function set compatible for downstream processing, we first standardize incomplete or ambiguous functions (*e.g.,* those missing explicit definitions in certain input or output parameters). We use a language model to infer the missing descriptions and types for each parameter $r \in \mathcal{X}_i \cup \mathcal{Y}_i$, denoted as $\mathrm{desc}(r)$ and $\mathrm{type}(r)$. Once the specifications are complete, we embed each parameter using a unified scheme. For each $r \in \mathcal{X}_i \cup \mathcal{Y}_i$, we concatenate its description and type with literal tags and encode them with an embedding model:

$$\mathbf{v}(r) = \phi\big(\texttt{DESC} \,\|\, \mathrm{desc}(r) \,\|\, \texttt{TYPE} \,\|\, \mathrm{type}(r)\big) \in \mathbb{R}^d.$$

3.1.2 Random Walk with Function Graph

With the parameter embeddings in place, we build a function graph that captures relationships between functions. In this graph, each function is a node, and edges are established based on the semantic fit between outputs and inputs. Concretely, let $G = (V, E)$ be a directed graph with $V = \mathcal{F}$. For two distinct functions f_i and f_j, we measure how well an output of f_i can serve as an input to f_j by the

[*] https://huggingface.co/datasets/Salesforce/xlam-function-calling-60k

[†] https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2

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

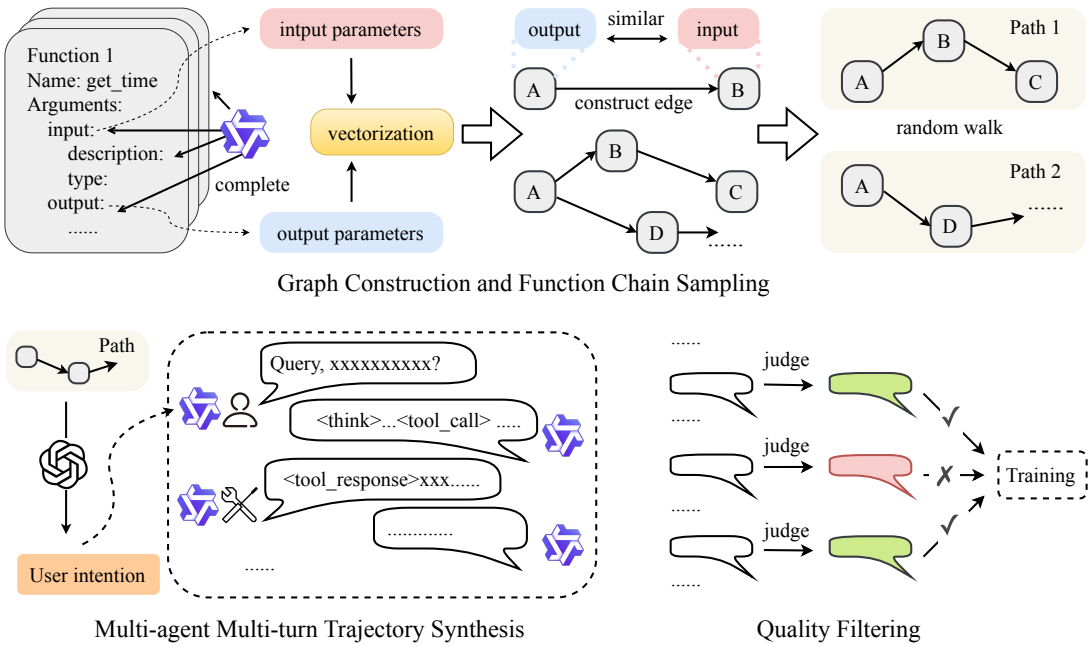

Figure 2: An illustration of the proposed data synthesis pipeline, with three key components: (1) Graph Construction and Function Chain Sampling, including function collection and refinement, function-graph construction, and random-walk sampling; (2) Multi-Agent Multi-Turn Trajectory Synthesis, using language models to simulate multiple roles and generate interactive trajectories; and (3) Quality Filtering, applying both trajectory-level and turn-level filtering to ensure high data quality.

maximum cosine similarity over their parameter embeddings:

$$s_{ij} = \max_{y \in \mathcal{Y}_i,\, x \in \mathcal{X}_j} \mathrm{sim}\big(\mathbf{v}(y), \mathbf{v}(x)\big),$$

where sim denotes cosine similarity. An edge $i \to j$ is created when this similarity exceeds a fixed threshold $\tau > 0$, indicating that an output parameter of f_i is semantically similar to an input parameter of f_j. To ensure the reliability of these connections, we further employ a language model as a validator and keep only the accepted links.

We also introduce randomness into edge construction. This yields a more diverse connectivity pattern that better reflects the real-world function usage. With the constructed graph, we sample function chains via random walks that traverse existing edges:

$$\mathcal{W} = (f_0, f_1, \ldots, f_L), L \sim \mathrm{Uniform}\{5, \ldots, 20\},$$

where $\mathcal{W}$ is a function chain, f_t denotes the function visited at step t, and L is the walking length. To prevent oversampling specific nodes, we impose a per-node visit limit during the random walk process. This encourages balanced graph coverage and mitigates bias toward high-degree hubs.

3.2 Multi-Agent Multi-Turn Trajectory Synthesis Framework

Based on the sampled function chains, we synthesize user intents to represent realistic user goals. A user intent represents the underlying goal that a user aims to achieve, rather than the exact query they pose. This distinction provides greater flexibility in modeling user behavior, as users often express the same intent in diverse ways. Since the tools are not executable in our setup, we employ a simulation framework to generate responses. The trajectories are created through a multi-agent simulation that involves three distinct agents:

- **User Agent**: Initiates the interaction by asking questions based on synthesized intent and drives the conversation forward.
- **Assistant Agent**: Responds to the user's queries, provides guidance, asks for clarification, or calls external functions when needed.
- **Tool Agent**: Simulates the execution of a function invoked by the assistant, returning tool responses based on simulated outputs.

These three agents operate in an iterative loop: the user agent poses a query, the assistant agent responds, and the tool agent simulates feedback

when required. The process continues until a predefined turn limit or stop condition is reached. This simulation produces a diverse set of interaction trajectories that serve as training data for modeling rich user–assistant–tool dynamics.

Empirical results demonstrate that even without real-world interactions, the large-scale synthetic data generated from these simulations significantly improves model performance. The flexibility and diversity of the simulated conversations provide valuable insights for training models in complex, multi-agent systems. For a detailed example of the generated data, see Appendix C.

3.3 Quality Filtering

To ensure reliable learning signals from synthesized interactions, we apply a two-stage filtering process: trajectory-level filtering to maintain goal alignment and coherence, followed by turn-level filtering to remove erroneous or misaligned steps.

3.3.1 Trajectory-Level Filtering

Given the predefined user intents (from the previous section), the user agent first determines whether the task has been completed during multi-turn interactions. Then, another LLM evaluates goal coverage, coherence, and cross-turn relevance. Only trajectories that align with the intent and exhibit a coherent flow are retained, while the rest are discarded. This stage produces a high-quality pool of trajectories for finer-grained filtering.

3.3.2 Turn-Level Filtering

Since trajectory-level filtering operates at a coarse granularity, it cannot effectively identify erroneous turns within otherwise coherent trajectories. Trajectories may still contain erroneous or misaligned turns—such as failed intermediate attempts, tool calls based on incorrect assumptions, or role drift in user/tool messages. We further refine the data by applying a separate LLM-based judge to each turn. The judge then masks invalid or role-inconsistent steps and retains only valid and coherent action steps, ensuring that the final training data supports effective problem solving and precise tool use.

3.4 Data Statistics

We split each trajectory into multiple samples using the turns that passed the turn-level quality filter and analyze both *trajectories* (orange) and *post-split samples* (blue). See the distribution analysis in Figure 3. From a trajectory-level perspective, the overall distribution of data lengths is relatively uniform, and users show strong participation throughout the dialogues. After the conversations are segmented into turns and filters are applied, the distribution skews toward shorter instances, which is expected. We further analyze the distribution of step counts for consecutive tool calls. A sizable fraction of turns proceed without any tool calls, likely because the synthetic user is prompted to mimic natural dialogue—holding back some constraints so that the model must ask follow-up questions and confirm details. Taken together, these findings indicate that current tasks leave room for greater complexity, motivating future work on more demanding tool-use scenarios.

In addition, we perform domain classification and statistical analysis of user intents throughout the synthesis process. Because each intent is composed via a function chain, a single intent can have multiple domain labels. As Figure 4 shows, the data analysis and entertainment domains are the most prevalent, while the remaining domains are relatively balanced.

4 Experiments

This section presents the performance of models of varying sizes after supervised fine-tuning (SFT) on our dataset. We first present the fine-tuning setup and then evaluate the resulting models against the baselines on widely-used tool-calling benchmarks.

4.1 Experimental Setup

Datasets. The ToolMind dataset comprises two components: (i) **ToolMind-Synthetic**: synthesized trajectories described in Section 3 and (ii) **ToolMind-Refined**: data augmented from open-source datasets, including the following datasets: xlam-function-calling-60k (Liu et al.), When2Call (Ross et al., 2025), glaive-function-calling-v2 [§], ToolACE (Liu et al., 2024b), BUTTONInstruct (Chen et al., 2025b), APIGen-MT-5k (Prabhakar et al., 2025), and the Tau-bench training set (Yao et al., 2024). All these datasets undergo the same split and quality-filtering procedures as the synthesized data. The statistics of the datasets can be found in Table 2. After preprocessing, each multi-turn conversation is split into multiple samples at each assistant message. For each sample, we retain the full prior context up to

[§] https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2

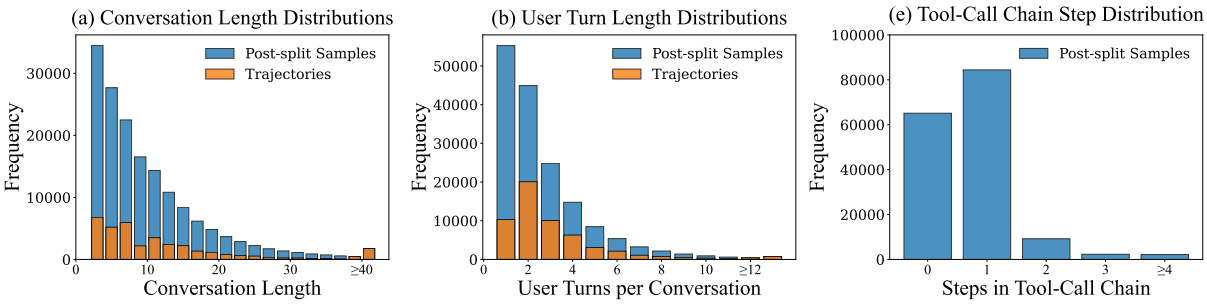

Figure 3: Distribution analysis of the proposed synthetic dataset. The three subplots present the distributions of conversation length, user turns per conversation, and tool-call chain steps, highlighting key statistical characteristics of the dataset and comparing the distributions between post-split samples and original trajectories.

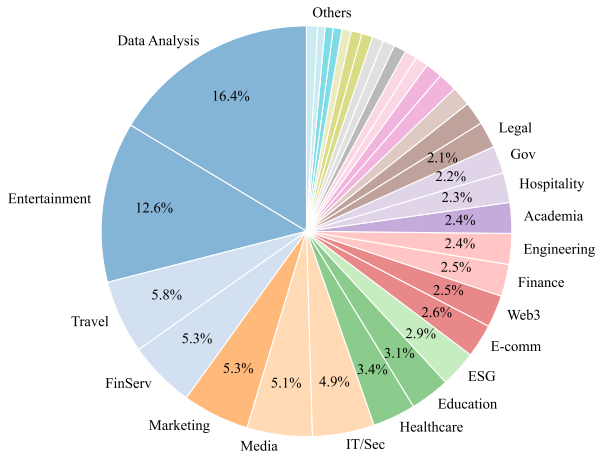

Figure 4: Distribution of User Intent Domains (low-frequency domains $\leq 2\%$ grouped as "others")

Dataset	# Trajectories (Queries)	# Samples after processing
xlam-function-calling-60k	60,000	101,363
When2Call (sft)	15,000	17,531
Glaive-function-calling-v2	112,960	20,017
ToolACE	11,300	7,327
BUTTONInstruct	8,000	21,202
APIGen-MT-5k	5,000	25,109
Tau-bench train set	–	12,882
ToolMind-Synthetic	40,161	163,180
ToolMind	111,941	368,611

Table 2: Statistical Overview of Data Samples. "Trajectories (Queries)" denote interaction sequences collected from datasets, while "Samples after processing" refer to samples after reconstruction and quality filtering.

and including the anchor assistant message and discard all subsequent turns. Each assistant response is structured into three parts: think, content, and tool calls. During training, loss is computed only on the tokens of the anchored assistant response.

Benchmarks. We evaluate our models on three widely used tool-calling benchmarks: BFCL-v4 (Patil et al.), τ-bench (Yao et al., 2024), and τ^2-bench (Barres et al., 2025). For BFCL-v4, we follow the official evaluation setup for Qwen3 and benchmark our model against larger-scale models on the official leaderboards as of October 2025. For a comprehensive comparison, we incorporate results from both open-source models `DeepSeek-V3` (Liu et al., 2024a), `DeepSeek-R1-0528` (Guo et al., 2025a), `Qwen3-235-instruct` (Yang et al., 2025a), `Kimi-K2-Instruct` (Team et al., 2025) and closed-source models `GPT-4o-2024-11-20`, `GPT5-2025-0807`, and `Gemini2.5-Pro` (Comanici et al., 2025). For τ-bench and τ^2-bench, we use GPT-4o as a user simulator and evaluate performance by directly comparing the model before and after fine-tuning.

Implementation Details. We perform supervised fine-tuning (SFT) on `Qwen3-8B`, and `Qwen3-14B` (Yang et al., 2025a). Training is conducted using the OpenRLHF (Hu et al., 2024) framework with a sequence length of 64k, a global batch size of 64, a learning rate of 5e-6, and a warmup ratio of 0.03. We train models on the entire dataset for two epochs.

4.2 Overall Performance

Berkeley Function-Calling Benchmark. We evaluate the models on the BFCL v4 benchmark (as of August 2025). The results are shown in Table 3. ToolMind consistently improves the tool-use capability of mid-sized models. On Qwen3-8B, ToolMind increases the overall score from 42.21 to 46.92 (+4.69%), with notable gains in *Multi Turn* (+7.74) and *Agentic Search* (+11.50). Similarly, on Qwen3-14B, ToolMind improves the overall score

Model	Overall	Single Turn		Multi Turn	Agentic	
		Non-live(AST)	Live(AST)		Search	Memory
GLM-4.5 (FC)	70.85	86.60	81.72	65.62	79.00	50.75
Claude-Opus-4-1-20250805 (FC)	70.36	88.38	81.50	57.88	77.00	62.15
Grok-4-0709 (Prompt)	61.60	81.27	69.73	43.25	72.00	54.41
GPT5-2025-0807 (FC)	59.22	72.92	58.25	28.50	84.50	57.63
Kimi-K2-Instruct (FC)	56.07	84.02	77.57	48.75	59.00	25.16
Qwen3-235-instruct (FC)	54.37	88.10	82.61	44.50	49.00	29.25
Gemini2.5-Pro (Prompt)	54.14	89.54	76.83	30.62	66.50	31.61
GPT-4o-2024-11-20 (FC)	50.27	83.88	70.54	42.50	40.50	28.82
DeepSeek-R1-0528 (FC)	48.97	75.73	80.90	44.50	63.00	0.00
xLAM-2-8b-fc-r (FC)	46.56	84.40	67.51	69.12	6.00	15.70
DeepSeek-v3 (FC)	45.20	88.77	79.94	33.00	32.50	22.37
ToolACE-2-8B (FC)	42.58	87.87	77.20	37.00	9.00	20.65
watt-tool-8B (FC)	38.69	87.54	75.87	37.88	4.00	8.60
Our Method						
Qwen3-8b (FC)	42.21	**88.27**	80.83	38.88	10.00	18.71
with ToolMind	**46.92**$^{+4.71}$	88.06	**81.42**$^{+0.59}$	**46.62**$^{+7.74}$	**21.50**$^{+11.50}$	**20.43**$^{+1.72}$
Qwen3-14b (FC)	45.14	**90.10**	**80.90**	44.12	12.50	**21.29**
with ToolMind	**50.54**$^{+5.40}$	89.00	80.83	**51.00**$^{+6.88}$	**35.50**$^{+23.00}$	17.85

Table 3: Overall Performance Comparison on BFCL-v4. The term "FC" denotes native support for function/tool calling, whereas "Prompt" refers to a walk-around implementation that leverages the model's inherent text generation ability. Specifically, the upper section of the table reports results from the BFCL August leaderboard.

Model	τ-bench			τ^2-bench			
	Avg.	retail	airline	Avg.	retail	airline	telecom
qwen3-8b (FC)	35.83	35.65	36.00	34.64	43.86	32.00	28.07
with ToolMind	**46.70**$^{+10.87}$	**57.39**$^{+21.74}$	**36.00**	**46.41**$^{+11.77}$	**59.65**$^{+15.79}$	**48.00**$^{+16.00}$	**31.58**$^{+3.51}$
qwen3-14b (FC)	38.78	49.56	28.00	40.65	52.63	36.00	**33.33**
with ToolMind	**53.00**$^{+14.22}$	**60.00**$^{+10.44}$	**46.00**$^{+18.00}$	**49.08**$^{+8.43}$	**59.65**$^{+7.02}$	**56.00**$^{+20.00}$	31.58

Table 4: Overall Performance Comparison on τ-bench and τ^2-bench. The table reports average scores (Avg.) and per-domain results across multiple domains (retail, airline, and telecom). For each base model, we compare the original model with the version fine-tuned on the ToolMind dataset.

from 45.14 to 50.54 (+5.40%), with substantial improvements in *Multi Turn* and *Search*.

Performance on *Single Turn* tasks remains largely unchanged after applying ToolMind. This does not necessarily indicate weaker training data; rather, it suggests that the benchmark may already be close to saturation, as even the strongest models achieve scores below 90 on this benchmark. In contrast, larger gains are observed in *Multi Turn* and *Agentic* settings.

τ-bench and τ^2-bench. Table 4 presents the performance comparison on τ-bench and τ^2-bench. The results show that ToolMind consistently improves the performance of Qwen3-8B and Qwen3-14B across different domains. On τ-bench, ToolMind increases the average score of Qwen3-8B from 35.83 to 46.70 (+10.87%), and improves Qwen3-14B from 38.78 to 53.00 (+14.22%). Similar gains are observed on the more challenging τ^2-bench, where ToolMind raises the average score from 34.64 to 46.41 for Qwen3-8B and from 40.65 to 49.08 for Qwen3-14B.

These results indicate that ToolMind effectively strengthens the model's capability in realistic multi-step decision-making scenarios. Compared to BFCL, these benchmarks include user interactions in the evaluation loop, making them closer to real-world agentic scenarios.

4.3 Ablation study

To evaluate the effectiveness of each component in ToolMind, we conduct an ablation study on Qwen3-8B. The study comprises the following configura-

Model	τ-bench			τ^2-bench				BFCL-v4
	Avg.	retail	airline	Avg.	retail	airline	telecom	overall
Qwen3-8b (FC)	35.83	35.65	36.00	34.64	43.86	32.00	28.07	42.21
with (a) ToolMind-Synthetic	42.31	42.61	42.00	38.85	42.98	42.00	**31.58**	46.87
with (b) ToolMind-Refined	**48.65**	51.30	**46.00**	42.16	57.89	44.00	24.56	45.88
with (c) no turn-level filtering	35.31	42.61	28.00	41.73	47.37	48.00	29.82	44.11
with ToolMind	46.70	**57.39**	36.00	**46.41**	**59.65**	**48.00**	**31.58**	**46.92**

Table 5: Ablation Study of ToolMind Components. Models trained on three data configurations are compared: (a) ToolMind-Synthetic: synthesized data only, (b) ToolMind-Refined: data augmented from open-source datasets, and (c) ToolMind without turn-level quality filtering.

Data Portion	0%	20%	60%	100%
BFCL-v4	42.21	44.70	45.96	46.92

Table 6: Impact of ToolMind training data scale on model performance. We report BFCL-v4 scores when fine-tuning the model with different portions of the ToolMind dataset (0%, 20%, 60%, and 100%).

tions: (a) ToolMind-Synthetic: using only synthesized data, (b) ToolMind-Refined: using only data augmented from open-source datasets. (c) ToolMind without turn-level quality filtering. The experimental results, shown in Table 5, demonstrate that the open-source augmented data contribute to significant performance improvements on τ-bench and τ^2-bench, while the graph sampling-based synthetic data lead to major gains on BFCL-v4, and the turn-level quality filtering significantly enhances the model performance. The model trained on ToolMind achieves the best results across nearly all evaluation metrics, demonstrating that each component of ToolMind is integral and effective.

4.4 Discussion

Data Scaling. We further investigate how model performance scales with the size of the training data. As shown in Table 6, performance improves consistently as the training data increases from 20% to 60%, and further to the full dataset. This indicates that the model continues to benefit from additional training data, highlighting the effectiveness and scalability of our dataset. Even with only 20% of the data, the model achieves competitive performance, demonstrating dataset quality.

Out-of-Distribution Generalization. We collect all tools used in the three evaluation benchmarks and perform fuzzy string matching to identify their occurrences in our dataset. We then compute the proportion of evaluation tools covered

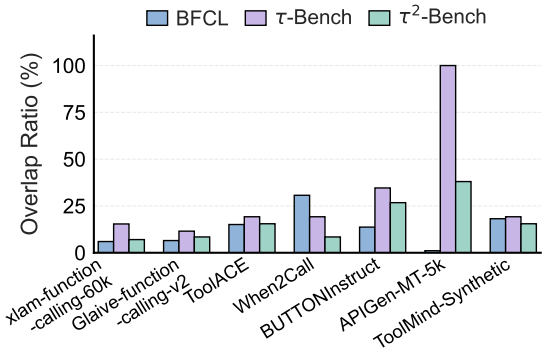

Figure 5: Tool overlap ratios between training datasets and evaluation benchmarks (BFCL-v4, τ-bench, and τ^2-bench), measured by fuzzy string matching.

by the dataset. As shown in Figure 5, the overlap ratios are generally low across most benchmarks. Even under fuzzy string matching, less than 20% of the evaluation tools overlap with those in ToolMind-Synthetic, indicating that the experiments of ToolMind-Synthetic are conducted under a largely **out-of-distribution** (OOD) setting. Notably, APIGen-MT-5k exhibits a substantially higher overlap ratio with τ-bench. This is likely because these datasets directly use tools in BFCL or τ-bench for data synthesis.

5 Conclusion

This paper presents ToolMind, a large-scale dataset designed to address data scarcity in tool-use for Large Language Models. Built through graph-based function sampling and multi-agent trajectory synthesis, ToolMind offers diverse, realistic function-calling trajectories with explicit reasoning traces. Experiments demonstrate that models fine-tuned on ToolMind achieve consistent and significant improvements across multiple benchmarks. Our work highlights the value of high-quality synthetic data for advancing tool-use capabilities and provides a valuable resource for future research.

Limitations

Beyond the substantial improvements demonstrated by ToolMind, there is still room for enhancing the complexity of multi-turn interactions, as certain real-world edge cases remain underrepresented. Additionally, while the dataset shows promising performance across various benchmarks, a more comprehensive evaluation across diverse tool types, user behaviors, and real-world scenarios is needed to better understand the dataset's generalization capabilities.

References

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. tau^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Amartya Chakraborty, Paresh Dashore, Nadia Bathaee, Anmol Jain, Anirban Das, Shi-Xiong Zhang, Sambit Sahu, Milind Naphade, and Genta Indra Winata. 2025. T1: A tool-oriented conversational dataset for multi-turn agentic planning. *arXiv preprint arXiv:2505.16986*.

Jinyang Chen, Haolun Wu, Jianhong Pang, Yihua Wang, Dell Zhang, and Changzhi Sun. 2025a. Tool learning with language models: a comprehensive survey of methods, pipelines, and benchmarks. *Vicinagearth*, 2(1):16.

Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. 2025b. Facilitating Multi-turn Function Calling for LLMs via Compositional Instruction Tuning. *arXiv preprint*. ArXiv:2410.12952 [cs].

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Bowen Fang, Wen Ye, Yunyue Su, Jinghao Zhang, Qiang Liu, Yesheng Liu, Xin Sun, Shu Wu, Jiabing Yang, Baole Wei, et al. 2026. Toolweaver: Weaving collaborative semantics for scalable tool use in large language models. *arXiv preprint arXiv:2601.21947*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025a. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

Ruixiang Feng, Zhenwei An, Yuntao Wen, Ran Le, Yiming Jia, Chen Yang, Zongchao Chen, Lisi Chen, Shen Gao, Shuo Shang, et al. 2025b. Cosineverifier: Tool-augmented answer verification for computation-oriented scientific questions. *arXiv preprint arXiv:2512.01224*.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*.

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

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2025b. StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models. *arXiv preprint*. ArXiv:2403.07714 [cs].

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Zilin Zhu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, et al. 2024. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*.

Chengrui Huang, Shen Gao, Zhengliang Shi, Dongsheng Wang, and Shuo Shang. 2025. Ttpa: Token-level tool-use preference alignment training framework with fine-grained evaluation. *arXiv preprint arXiv:2505.20016*, 10.

Chengrui Huang, Zhengliang Shi, Yuntao Wen, Xiuying Chen, Peng Han, Shen Gao, and Shuo Shang. 2024. What affects the stability of tool learning? an empirical study on the robustness of tool learning frameworks. *arXiv preprint arXiv:2407.03007*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023a. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. *arXiv preprint*. ArXiv:2304.08244 [cs].

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

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. *arXiv preprint*. ArXiv:2410.04587 [cs].

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

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, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. 2024b. ToolACE: Winning the Points of LLM Function Calling. *arXiv preprint*. ArXiv:2409.00920 [cs].

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated PIpeline for Generating Verifiable and Diverse Function-Calling Datasets.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. APIGen-MT: Agentic Pipeline for Multi-Turn Data Generation via Simulated Agent-Human Interplay. *arXiv preprint*. ArXiv:2504.03601 [cs].

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

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. 2025b. Smart: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4604–4621.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv preprint*. ArXiv:2307.16789 [cs].

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

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. 2025. When2Call: When (not) to Call Tools. 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 3391–3409, Albuquerque, New Mexico. Association for Computational Linguistics.

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

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. 2025. Toolflow: Boosting llm tool-calling through natural and coherent dialogue synthesis. 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 4246–4263.

Chunyu Wei, Wenji Hu, Xingjia Hao, Xin Wang, Yifan Yang, Yueguo Chen, Yang Tian, and Yunhai Wang. 2025. Graphchain: Large language models for large-scale graph analysis via tool chaining. *arXiv preprint arXiv:2511.00457*.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. 2024. AgentGym: Evolving Large Language Model-based Agents across Diverse Environments. *arXiv preprint*. ArXiv:2406.04151 [cs].

Weikai Xu, Chengrui Huang, Shen Gao, and Shuo Shang. 2025a. Llm-based agents for tool learning: A survey: W. xu et al. *Data Science and Engineering*, pages 1–31.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. 2025b. TOUCAN: Synthesizing 1.5M Tool-Agentic Data from Real-World MCP Environments. *arXiv preprint*. ArXiv:2510.01179 [cs].

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

Chen Yang, Guangyue Peng, Jiaying Zhu, Ran Le, Ruixiang Feng, Tao Zhang, Wei Ruan, Xiaoqi Liu, Xiaoxue Cheng, Xiyun Xu, et al. 2025b. Nanbeige4-3b technical report: Exploring the frontier of small language models. *arXiv preprint arXiv:2512.06266*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint*. ArXiv:2406.12045 [cs].

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Yirong Zeng, Xiao Ding, Yutai Hou, Yuxian Wang, Li Du, Juyi Dai, Qiuyang Ding, Duyu Tang, Dandan Tu, Weiwen Liu, et al. 2025b. Tool zero: Training tool-augmented llms via pure rl from scratch. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 9135–9147.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

A Training Details

This section details the training parameter of Qwen3-8b and Qwen3-14b on ToolMind.

Parameter	Value
BF16	True
Learning Rate	5×10^{-6}
Batch Size	64
LR Scheduler Type	cosine
Maximum Sequence Length	65536
Training Epochs	2
Use Liger Kernel	True
Warmup Ratio	0.03
L2	0.01

Table 7: Training hyperparameters and optimization settings for Qwen3-8B and Qwen3-14B on ToolMind.

B Prompts

B.1 Prompt for completing functions

This section details the prompt for completing functions and determining correlations.

B.1.1 Prompt for Function Input Prediction

Please help me predict the input of the function.

Return only one result in JSON format with two fields: input description and input structure. Input description should describe the content of the input, while input structure should be a list of parameters, each with a name, a description and a type.

The function is: {function}

B.1.2 Prompt for Function Output Prediction

Please help me predict the output of this function.

Return only one result in JSON format with two fields: output description and output structure. Output description should describe the content of the output, while output structure should be a list of parameters, each with a name, a description and a type.

The function is: {function}

B.1.3 Prompt for Function Association Evaluation

Please determine whether the following "source_function" and "target_function" can be highly correlated or the output result of "source_function" is suitable as the input parameter of "target_function". If the correlation is large or suitable for parameter passing, the specific function is as follows:

Source_function:

{source_function}

Target_function:

{target_function}

Please evaluate the strength of the association edges and assign a score from 0 to 9 for the following aspects: field transitivity, coherence of potential user intent paths. The output should be a JSON object with exactly two fields: "Field transitivity", "Potential user intent path coherence". Do not output anything else!!!

B.2 Prompt for trajectory synthesis

This section details the prompts for trajectory synthesis, including prompts for user and tool simulator.

B.2.1 Prompt for synthesis user intent

Suppose you have another assistant who has access to the following tools to get information. Please generate one task instruction that mimic real human users and their intentions, such as having different personalities and goals. Note that the intent should be as natural as possible, covering as many tools as possible, but not forcing overwriting if the tools are not closely related. Please ignore image-related tools and do not generate image-related instructions. User intents should be highly consistent, avoiding the awkward patchwork of several unrelated tasks.

Tools

{tools}

Please output the results strictly in JSON format with two fields: "Task Instruction" and "Tool Usage" and don't output anything else!!

B.2.2 Prompt for user simulator

You are a human user and must act as a genuine user throughout the conversation, interacting in a manner consistent with normal human behavior. Your primary goal is to achieve the following intent by seeking guidance, advice, or assistance from other participants.

Your intent:

{intent}

Please adhere strictly to the following guidelines:

1. Role Consistency and Natural Interaction: Always maintain the role of a user. Do not respond as an assistant, AI, or any authoritative figure. Speak naturally, as a real human would. Avoid repetitive, mechanical, or overly structured responses.

2. Incremental Disclosure: Do not reveal your entire intent at once. Unfold your needs gradually over multiple turns. Use common human conversation strategies, such as showing uncertainty when appropriate.

3. Response to Fulfillment: If the the other participant successfully fulfills your intent, output '###STOP###' immediately. Do not output '###STOP###' in the first turn, regardless of the conversation!!

B.2.3 Prompt for tool simulator

You are simulating a high-performance computer system with complete computational capabilities. You have access to extensive external knowledge, can execute any arbitrary function, and operate without errors. For a given function, you should simulate the execution of a computer system program as accurately as possible.

Function info

{function_info}

Function call

{tool_calls}

Given this function call, you should execute the function and return the results strictly in JSON format. Your response should contain only the JSON result, without any additional or irrelevant text. The result must be enclosed within <func_return> and <func_return> tags. If the function call is invalid (e.g., incorrect function name, missing or malformed arguments), return a JSON error message clearly indicating the cause.

Example of function call and function return:

["name": "get_weather", "arguments": "city": "New York"]

<func_return> "temperature": "25°C" <func_return>

B.3 Prompt for quality filtering

This section details the prompts used for quality filtering, including trajectory-level filtering and turn-level filtering.

B.3.1 Prompt for trajectory-level quality filtering

Please strictly evaluate the quality of the following multi-turn dialogue data based on the following criteria: contextual coherence, role consistency, logical soundness, and accuracy of tool usage. Your task is to make a binary judgment — if the dialogue is of good quality, output 1; otherwise, output 0.

Tools

{tools}

###Multi-turn conversations

{messages}

Please make a strict and comprehensive assessment of the dialogue's quality, considering whether it maintains contextual coherence, consistent role behavior, logical reasoning, and correct use of tools. Finally, output only a single digit: 0 or 1. Do not include any other text or explanation.

B.3.2 Prompt for turn-level quality filtering

Please strictly evaluate the quality of the last response in the following dialogue data, based on contextual coherence, logical consistency, and accuracy of tool usage. Determine whether the response is semantically aligned with the previous dialogue, logically sound without contradictions, and employs the mentioned tools

correctly according to their definitions and argument structures. If the response is of good quality, output 1; otherwise, output 0.

Tools mentioned in the conversation

{tools}

Conversation history

{messages}

Last response

{response}

Please make a strict judgment on whether the last response is of good or poor quality, considering contextual coherence, logical soundness, and correctness of tool usage. Finally, output only a single digit: 0 or 1. Do not output any other text or explanation.

C Case Study

Function Chains

{'name': 'getStockLocations', 'description': 'Retrieves a list of stock locations for an eCommerce application.', 'parameters': {'type': 'dict', 'properties': {'limit': {'description': 'Limits the number of items on a page (max 100).', 'type': 'float'}}, 'required': []}, 'required': None}

{'name': 'checkInventory', 'description': 'Check the inventory of a specific product', 'parameters': {'type': 'dict', 'properties': {'product_code': {'type': 'string', 'description': 'The code of the product to check inventory for (e.g., ABC123)'}, 'location': {'type': 'string', 'description': 'The location to check inventory at (e.g., warehouse A, store B)'}}, 'required': ['product_code']}, 'required': None}

{'name': 'checkIpAddress', 'description': 'Check if an IP address is safe or not.', 'parameters': {'type': 'dict', 'properties': {'ip_address': {'type': 'string', 'description': 'The IP address to be checked for safety'}}, 'required': ['ip_address']}, 'required': None}

User Intent

I'm a warehouse manager for an eCommerce company. First, I need to check where our inventory is stored by retrieving the stock locations. Then, I want to verify the availability of product 'XYZ789' specifically at 'warehouse C' to plan restocking. Finally, I need to ensure our system's security by checking if the IP address 192.168.1.100 is safe.

Trajectory

user: Can you retrieve the current stock locations for me?

assistant: <think>Okay, the user is asking to retrieve the current stock locations. Let me check the available tools. There's a function called getStockLocations. The parameters for that include a limit, which is optional and can be up to 100. The user didn't specify a limit, so I don't need to include that. Since the function doesn't require any parameters, I can call it without any arguments. I should generate the tool call with the name getStockLocations and empty arguments. Wait, but the parameters are a dict, so maybe I should pass an empty object. Yeah, that makes sense. So the tool call would be "name": "getStockLocations", "arguments": . I need to make sure that the JSON is correctly formatted. No other parameters are required, so that's all.</think> <tool_call> {"name":"getStockLocations", "arguments":"{}"} </tool_call>

tool: { "stock_locations": ["id": 1, "name": "Warehouse A", "address": "123 Main St, Cityville", "capacity": 5000 , "id": 2, "name": "Store B", "address": "456 Oak Ave, Townsburg", "capacity": 2000 , "id": 3, "name": "Distribution Center C", "address": "789 Pine Rd, Countryside", "capacity": 10000], "pagination": "page": 1, "total_pages": 5 }

assistant: <think>Okay, let me process the user's request. They asked for the current stock locations, so I called the getStockLocations function. The response came back with three locations: Warehouse A, Store B, and Distribution Center C. Each has an ID, name, address, and capacity. The pagination shows there are 5 total pages, but the user might not need all of them right now. I should present this information clearly. Start by listing each location with their details. Maybe use bullet points for readability. Also, mention the pagination part so the user knows there's more data available if needed. They might want to ask for the next page or check a specific prod-

uct's inventory next. I'll keep the tone friendly and offer further assistance in case they need more details or another action, like checking inventory for a product. Make sure to highlight the key points like names, addresses, and capacities. Alright, let's put that together in a concise response.</think> Here are the current stock locations retrieved:1. **Warehouse A** - **Address:** 123 Main St, Cityville - **Capacity:** 5,000 units 2. **Store B** - **Address:** 456 Oak Ave, Townsburg - **Capacity:** 2,000 units 3. **Distribution Center C** - **Address:** 789 Pine Rd, Countryside - **Capacity:** 10,000 units There are **5 total pages** of results (you're currently viewing page 1). Would you like to: - Check inventory for a specific product at one of these locations? - Retrieve the next page of stock locations?

user: Can you check the availability of product XYZ789 at Distribution Center C?

assistant: <think>Okay, the user is asking to check the availability of product XYZ789 at Distribution Center C. Let me see which tool to use here.Looking at the available tools, there's the checkInventory function. It requires the product_code and optionally the location. The user specified both the product code and the location, so I need to call checkInventory with those parameters.Product code is XYZ789, and the location is Distribution Center C. Let me make sure the parameters are correctly formatted. The location in the tool might need to be a string, so I'll input "Distribution Center C" as the location. I should structure the tool call with these arguments. Let me double-check if there are any other required parameters. The tool's required field is product_code, which is provided. Location is optional but the user specified it, so include it.Alright, the tool call should be correct with those parameters.</think> <tool_call> {"name": "checkInventory", "arguments": "{"product_code": "XYZ789"}} </tool_call>

tool: {"product_code": "XYZ789", "location": "Distribution Center C", "available_quantity": 150, "last_updated": "2023-10-05T14:30:00Z" }

assistant: <think>Okay, let me process the user's request. They wanted to check the availability of product XYZ789 at Distribution Center C. I called the checkInventory tool with the product code and location. The response shows 150 units available, last updated on October 5th.Now, I need to present this information clearly. The user might be a warehouse manager or someone involved in order fulfillment. They probably need this info to fulfill an order or manage stock levels. I should mention the available quantity and the last updated timestamp to indicate the data's freshness. Also, offer further assistance in case they need more details or actions, like checking another location or placing an order. Keep the response concise but informative.</think> Here is the inventory availability for product **XYZ789** at **Distribution Center C**:- **Available Quantity:** 150 units - **Last Updated:** October 5, 2023, 2:30 PM UTC Would you like to check availability at another location or proceed with an order?

user: Is the IP address 192.168.1.100 flagged as a security risk in our system?

assistant: <think> Okay, the user is asking if the IP address 192.168.1.100 is flagged as a security risk in our system. Let me check the tools available. There's a function called checkIpAddress that takes an IP address as a parameter. I need to call that.First, I'll verify the parameters. The function requires the ip_address field, which the user provided. The IP given is 192.168.1.100. I should make sure it's a valid IP. Since it's in the correct format, I can proceed. I don't see any other parameters needed for this function. So, I'll structure the tool call with the name checkIpAddress and the ip_address argument. No other tools are relevant here, like getStockLocations or checkInventory. Just need to output the tool call in the specified XML format. Let me double-check the syntax to avoid any errors. The JSON should have the name and arguments as specified. Alright, that should do it. </think> <tool_call> {"name": "checkIpAddress", "arguments": "{"ip_address": "192.168.1.100"}"} </tool_call>

tool: {"status": "safe", "ip_address": "192.168.1.100", "reason": "Private IP address (RFC 1918 compliant)"}

assistant: <think>Okay, let me break this down. The user asked if the IP address 192.168.1.100 is flagged as a security risk. I called the checkIpAddress tool with that IP. The response says it's marked as safe because it's a private IP address under RFC 1918. Hmm, I need to explain this clearly. First, confirm the IP's status. Then, explain why it's safe. Maybe mention that private IPs are used internally and not routable on the public internet. Also, note that RFC 1918 defines these ranges. The user might not be familiar with RFC standards, so a brief explanation would help. Finally, offer further assistance if they need more checks. Keep it concise and friendly.</think> The IP address **192.168.1.100** is **not flagged as a security risk** in our system. It is categorized as a **private IP address** (per [RFC 1918](https://tools.ietf.org/html/rfc1918)), which means it is reserved for internal networks and cannot be routed over the public internet. These addresses are inherently non-routable and considered safe by design. Would you like to check another IP address or investigate further?

user: ###STOP###

794