

# Enhancing Tool Calling in LLMs with the International Tool Calling (ITC) Dataset

Anonymous ACL submission

## Abstract

Tool calling allows large language models (LLMs) to interact with external systems like APIs, enabling applications in customer support, data analysis, and dynamic content generation. Despite recent advances, challenges persist due to limited datasets with simulated or inaccessible APIs and insufficient geographic diversity. To address this, we present the International Tool Calling (ITC) dataset, designed for real-world, international tool calling scenarios. ITC includes 3,571 real APIs and 17,540 tool calling tasks across 20 categories and 40 countries. The dataset was constructed through a four-stage pipeline: API collection and construction, query generation, query scoring and filtering, and question-answer pair generation. Experiments reveal substantial performance gaps between open- and closed-source LLMs, while fine-tuning on ITC significantly improves generalization. ITC offers a valuable resource for advancing LLM capabilities in complex, multi-tool, and international contexts. Dataset: <https://anonymous.4open.science/r/International-tool-calling-ITC-dataset-5FD7/>.

## 1 Introduction

Tool calling empowers large language models (LLMs) to interact with external systems—such as databases, APIs, and software tools—extending their capabilities beyond text generation (Schick et al., 2023). By invoking tools, LLMs can access real-time data, perform complex computations, and execute actions beyond their training data (Nakano et al., 2021). This functionality is essential for tasks such as automated customer support, data analysis, and dynamic content generation, where external resource integration enhances both performance and utility. As surveyed in (Mialon et al., 2023), tool calling enables more sophisticated, context-aware interactions, making LLMs valuable across diverse domains.

Recent advances have led to the development of several datasets and benchmarks to improve tool-use capabilities in LLMs. Notable examples include API-BLEND (Basu et al., 2024), API-Gen (Liu et al., 2024c), and ToolACE (Liu et al., 2024b), which focus on API-based function calling across a variety of use cases. Others, such as Gorilla (Patil et al., 2023) and ToolLLM (Qin et al., 2023), address real-world tool invocation, reducing hallucinations and improving accuracy. More complex datasets like Seal-Tools (Wu et al., 2024), PLUTO (Huang et al., 2024), and SciTool-Bench (Ma et al., 2024) explore multi-step reasoning and domain-specific tool use. Collectively, these benchmarks have advanced the development of LLMs capable of interacting with external tools effectively.

However, key challenges remain. Many existing datasets rely on simulated APIs, lacking the complexity and variability of real-world tool usage (e.g., Seal-Tools (Wu et al., 2024)). Others use proprietary or inaccessible APIs, as in ToolLLM (Qin et al., 2023), limiting reproducibility and real-world deployment. Accessibility is further hindered by datasets that are not publicly available. Moreover, most existing benchmarks are US-centric, making them unsuitable for region-specific tasks in a global context. For example, while APIs like Yahoo\_Weather can retrieve data for major cities like Shenzhen, they often fail to provide detailed, district-level information (e.g., Nanshan), underscoring the need for broader geographic coverage and finer granularity.

To address these limitations, we introduce the International Tool Calling (ITC) dataset, designed for real-world, globally distributed tool calling scenarios. ITC is constructed through a four-stage pipeline: (1) **API Collection and Construction**, (2) **Query Generation**, (3) **Query Scoring and Filtering**, and (4) **QA Pair Generation**. The final dataset comprises 3,571 real-world APIs and

17,540 tool calling tasks—15,790 for training and 1,750 for testing—covering 20 categories across 40 countries. It includes 64.2% global APIs, which provide cross-country and multilingual support, as well as region-specific APIs, primarily from the US and China, along with long-tail APIs from over 27 additional countries. The dataset consists of 14,295 single-tool and 3,245 multi-tool tasks, with a deliberate focus on supporting underrepresented APIs, thereby enhancing its utility for diverse and internationally relevant tool calling applications.

We benchmarked 16 open-source and 8 closed-source LLMs on the ITC test set. Results revealed substantial performance gaps across multiple metrics, highlighting strengths and weaknesses in tool usage—especially in handling non-existent tools, missing parameters, and incorrect parameter generation. Furthermore, we fine-tuned several open-source LLMs (not previously trained on tool calling datasets) using ITC. These models exhibited significant performance improvements, demonstrating that ITC effectively enhances generalization and robustness in complex, real-world tool calling tasks.

Dataset	# Tools	Source	Access.	# Tasks	Ex.	P.L.
API-BLEND	199	Sim.	×	189,040	×	Eng.
APIGen	3,673	Real	×	60,000	✓	Eng.
Gorilla	1,645	Real	✓	16,450	✓	Eng.
Seal-Tools	4,076	Sim.	✓	14,076	×	Eng.
ToolACE	26,507	Sim.	×	11,300	×	Eng.
ToolBench	16,464	Real	✓	126,486	✓	Eng.
RoTBench	568	ToolEyes	×	105	✓	Eng.
MLLM-Tool	932	Real	×	11,642	✓	Eng.
PLUTO	2,032	Sim.	×	5,824	×	Eng.
SciToolBench	2,446	Sim.	×	856	×	Eng.
GeoLLM-QA	117	Real	×	1,000	×	Eng.
INJECAGENT	17	Sim.	✓	1,054	✓	Eng.
StableToolBench	16,464	ToolBench	✓	126,486	✓	Eng.
ToolEyes	568	Sim.	✓	382	✓	Eng.
ToolSword	100	Sim.	×	440	✓	Eng.
Hammer	-	APIGen	×	67,500	×	Eng.
<b>ours</b>	<b>3,573</b>	<b>Real</b>	<b>✓</b>	<b>17,540</b>	<b>✓</b>	<b>Multi</b>

Table 1: Summary of existing tool calling datasets. Access.: Public accessibility; Ex.: Executability of the APIs; P.L.: Primary language; Eng.: English; Sim.: Simulated APIs.

## 2 Related Work

Existing benchmarks designed to enhance the tool-invocation capabilities of large language models (LLMs) span a wide range of tasks, including API-based interactions, multi-modal tool usage, and robustness evaluations. Datasets such as API-BLEND (Basu et al., 2024), APIGen (Liu et al., 2024c), and ToolACE (Liu et al., 2024b) focus on API-based function calling, providing

diverse APIs for training and evaluation. While APIGen and ToolACE contain thousands of executable APIs, API-BLEND primarily supports semantic parsing and slot-filling tasks. Gorilla (Patil et al., 2023) and ToolLLM (Qin et al., 2023) enhance LLM capabilities in real-world API interactions, aiming to improve accuracy and reduce hallucinations. Meanwhile, Seal-Tools (Wu et al., 2024), PLUTO (Huang et al., 2024), and SciToolBench (Ma et al., 2024) introduce more complex tool-use scenarios, including multi-step reasoning and domain-specific applications. Other benchmarks, such as RoTBench (Ye et al., 2024c) and StableToolBench (Guo et al., 2024), assess LLM robustness and stability, while ToolEyes (Ye et al., 2024a) and ToolSword (Ye et al., 2024b) focus on cognitive abilities and safety in tool use. Additionally, multi-modal frameworks like MLLM-Tool (Wang et al., 2024) extend tool learning to non-text modalities, supporting interactions with images, text, and audio.

Table 1 provides an overview of representative tool calling datasets and frameworks. Despite notable progress, existing resources exhibit several critical limitations. Many rely on simulated APIs, which do not reflect the complexity and variability of real-world interactions. Datasets that incorporate real APIs often use proprietary or inaccessible endpoints, restricting their practical utility. Moreover, over half of the benchmarks are not publicly available, and even those that are—such as ToolBench—frequently include non-functional or outdated APIs. Finally, most datasets are heavily US-centric, limiting their relevance for tasks requiring region-specific information in a global context.

## 3 Dataset Curation

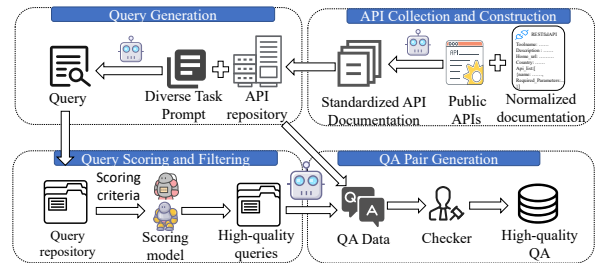


Figure 1: Dataset construction flowchart.

We construct our dataset using a four-stage pipeline designed for scalability and minimal human intervention. In **Stage 1: API Collection and Construction**, automated tools extract API

documentation. In **Stage 2: Query Generation**, GPT-4o generates detailed API instructions. **Stage 3: Query Scoring and Filtering** leverages Claude-3.5-Sonnet and Gemini-1.5-Pro to refine queries for clarity, relevance, and executability. Finally, in **Stage 4: QA Pair Generation**, GPT-4o produces high-quality question-answer pairs. This pipeline supports easy expansion to new APIs, as illustrated in Figure 1.

### 3.1 API Collection and Construction

**API Collection:** ToolBench (Qin et al., 2023) is a widely used benchmark for tool invocation, primarily sourcing APIs from RapidAPI. However, its coverage is largely US-centric, with limited representation from other regions. Many of the included APIs also require paid subscriptions, which restricts accessibility.

To overcome these limitations, we collected 49,937 real REST APIs across diverse domains such as social media, e-commerce, and weather. In addition to RapidAPI Hub<sup>1</sup>, we sourced APIs from international platforms including Juhe API<sup>2</sup> and XiaRou API<sup>3</sup> in China, as well as community-maintained repositories on GitHub like free-api<sup>4</sup> and public-apis<sup>5</sup>. These efforts aim to create a globally diverse and publicly accessible API resource. The collected APIs are organized into 20 functional categories, including Entertainment, Finance, and Education.

**API Supplementation:** Our collected APIs vary widely in documentation quality, with many missing parameter details, examples, or using inconsistent formats. To ensure reliable parsing by large language models, we apply a structured supplementation pipeline that standardizes all APIs into a uniform schema—including name, description, endpoint, HTTP method, authentication, input parameters (with types and constraints), and output format (see Figure 4 in Appendix A).

For incomplete or unclear documentation, we leverage GPT-4o with tailored prompts to auto-complete and refine API specs based on available metadata like endpoint names or sample inputs/outputs (example prompt in Figure 7). We then validate these by executing sample calls, correcting any inconsistencies or hallucinations man-

ually. APIs with irreparable documentation issues are discarded.

This process yields comprehensive, consistent API specifications that improve clarity and usability for both LLMs and developers (example in Figure 15, Appendix G).

**API Filtering:** The initial collection of 49,937 free APIs included a substantial number of redundant or low-quality entries. Many offered overlapping functionalities (e.g., weather forecasting, currency conversion), while others suffered from instability, outdated services, insufficient documentation, or inaccurate outputs.

To ensure the quality and reliability of our dataset, we implemented a two-stage filtering pipeline:

**Stage 1: Basic Functionality Screening** We developed automated scripts to issue predefined queries to each API and evaluated the responses. APIs were excluded if they: (1) failed to respond within a 10-second timeout, (2) returned HTTP error codes (e.g., 404, 500), (3) consistently produced empty or null responses, or (4) returned malformed or non-JSON outputs.

This initial screening eliminated the majority of non-functional or poorly maintained APIs, reducing the pool from 49,937 to 5,410. Representative failure cases are shown in Figures 5 and 6 in Appendix A.

**Stage 2: Quality-Oriented Filtering** The remaining 5,410 APIs were further evaluated for stability, update frequency, and response informativeness through a combination of automated tests and manual review. Stability was assessed by invoking each API 3–5 times over a 48-hour period; APIs exhibiting intermittent failures, highly variable latency (e.g., from 300 ms to over 5 seconds), or inconsistent response formats (such as changing data types or fields) were discarded. Update frequency was examined using version histories, metadata (e.g., last update timestamps), and external sources like changelogs and issue trackers. APIs with no updates for over 12 months or broken links were removed. Response informativeness was measured by submitting task-specific queries and evaluating the semantic richness of the outputs; APIs that returned minimal, placeholder, or heavily post-processed responses were excluded.

This filtering process eliminated 1,839 additional APIs, resulting in a curated set of 3,571 high-quality APIs—only 7.1% of the original 49,937.

<sup>1</sup><https://rapidapi.com/>

<sup>2</sup><https://www.juhe.cn/>

<sup>3</sup><https://api.aa1.cn/>

<sup>4</sup><https://github.com/fangzesheng/free-api>

<sup>5</sup><https://github.com/public-apis/public-apis>

These APIs are stable, actively maintained, and provide meaningful outputs, forming a reliable foundation for generating tool-use instructions.

### 3.2 Query Generation

To address the often-overlooked challenges of language and region specificity in tool calling, we design tasks focused on retrieving localized information through region-specific APIs. For instance, imagine a Japanese tourist planning a trip to Lijiang, China. They may require detailed weather updates and travel recommendations from both Japanese and Chinese APIs, with content presented in Japanese and Chinese, respectively. This scenario highlights the importance of enabling LLMs to perform cross-lingual reasoning and interact effectively with APIs across different linguistic and geographic contexts.

Based on the above principles, we curated 36 seed examples across a range of scenarios. For each, we selected one or more APIs and generated three sample queries using the following strategies to ensure quality and diversity:

1. **Biased Sampling Correction:** Our API collection follows a long-tail distribution, with countries like China and the U.S. contributing disproportionately more APIs. To ensure geographic diversity, we over-sample APIs from underrepresented countries, reducing bias and enhancing cross-regional representation.
2. **Tool Confusion Strategy:** To increase task complexity and test model robustness, we include similar APIs within the same functional category or across countries. Queries are designed to be intentionally ambiguous, forcing the model to distinguish between APIs with overlapping capabilities—thus evaluating its ability to resolve fine-grained distinctions.

We used GPT-4o to generate queries based on these examples, producing diverse outputs formatted in JSON. Each entry includes a “Thought” field capturing the model’s reasoning process in the same language as the query, and an “Action” field specifying the corresponding API calls.

tool calling tasks can be broadly categorized into two types: **Single Tool Calling Tasks**, which involve invoking a single API to complete a task, and **Multiple Tools Calling Tasks**, which require coordinating multiple API calls, potentially across

different domains or countries. Multiple Tools Calling Tasks can be further classified into three subtypes. In **Repeated Tool Calling**, the same API is invoked multiple times with different parameters to complete a multi-stage objective. **Parallel Tool Calling** involves invoking two or more APIs simultaneously to address different components of the task. In contrast, **Nested Tool Calling** requires a step-by-step process where the output of one API serves as the input for the next, forming a sequential chain of reasoning.

### 3.3 Query Scoring and Filtering

In the last step, we obtained 44,198 queries. However, many of these queries presented issues such as unclear requirements, insufficient relevance to the tools, non-standard language, and failure to appropriately adhere to cultural context.

Our query selection method consists of two main steps: Query Scoring and Query Selection. In the **Query Scoring** step, we addressed the limitations of existing datasets, which often lack fine-grained evaluation standards for assessing the quality of generated queries or question-and-answer pairs. Rather than relying on coarse metrics, we introduce five comprehensive scoring dimensions—Relevance, Practicality, Linguistic Applicability, Clarity, and Specificity (see Appendix F.1 for details) to ensure a thorough evaluation of instructions from multiple perspectives. To mitigate potential model bias, as highlighted in recent work (Zheng et al., 2023), we employ two independent scoring standards: Anthropic’s Claude-3.5-sonnet and Google’s Gemini-1.5-pro. Both models assign scores from 1 (very low quality) to 5 (highest quality). Filtering and optimization are performed based on the aggregated scores from both models.

During the **Query Filtering** step, we apply rigorous filtering criteria to retain only the highest-quality queries. Specifically, only instructions that receive a score higher than 4 from both the Claude and Gemini models are considered high-quality. This effectively excludes lower-scoring instructions that may contain irrelevant or poorly structured content. Through this stringent filtering process, we ensure that the retained queries meet high standards of relevance, practicality, clarity, specificity, and linguistic applicability, ultimately enhancing the reliability and usability of the dataset for downstream tasks.

Following scoring and filtering, the dataset was refined to a final set of 17,540 high-quality queries.



### 3.4 Question-and-Answer Pair Generation

The last step is to use GPT-4o model to generate answer for each query, along with relevant API information into the GPT-4o model. The model then generates the corresponding thought process and identifies the appropriate APIs to be called. This process requires careful handling of the model’s reasoning to ensure that both the generated answers and the API calls are contextually appropriate. Furthermore, ensuring the coherence and accuracy of the answers across multiple languages adds an additional layer of complexity, as it demands that the model appropriately handles language-specific features while maintaining high-quality outputs for a diverse set of use cases. For single-tool tasks, the complexity is relatively low, so we directly use prompt templates to generate question-and-answer pairs. However, for multiple tools calling tasks, in addition to using prompt templates, we employ another large language model as a Checker that has visibility into the entire generation process. This Checker validates the generated question-and-answer pairs, thereby enhancing their accuracy.

## 4 Data Statistics

Our International Tool Calling (ITC) dataset comprises 3,571 APIs and a total of 17,540 question-and-answer pairs, including 15,790 for training and 1,750 for testing. In the following sections, we detail the composition of the dataset from two perspectives: APIs and Tasks.

### 4.1 Statistics on APIs

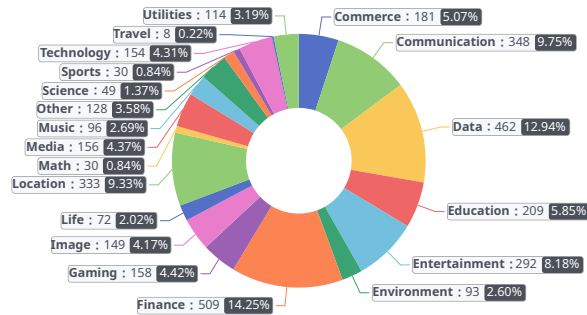


Figure 2: Distribution of APIs by Category in the ITC Dataset.

Figure 2 illustrates the distribution of APIs across 20 Category in the ITC Dataset. The largest categories are Finance (14.25%), Data (12.9%), Communication (9.75%), and Entertainment (8.18%). Conversely, the smallest categories

include Travel (0.22%), Math (0.84%), and Sports (0.84%).

Our dataset demonstrates notable geographical diversity, encompassing APIs from over 30 countries and regions. We classify these APIs into two categories: global APIs, which provide information across multiple countries and languages, such as machine translation and weather forecasting. These APIs are predominantly from the USA. The second category is country-specific APIs, which offer information tailored to a particular country and its language, such as local weather updates and news. Global APIs account for 2,291 samples (64.2%). China and the United States contribute the majority of country-specific APIs. Additionally, long-tail APIs, originating from over 27 other countries and regions, contribute 11.31% of the dataset. A detailed breakdown of the country distribution can be found in Figure 8 in Appendix A.5

### 4.2 Statistics on Tasks

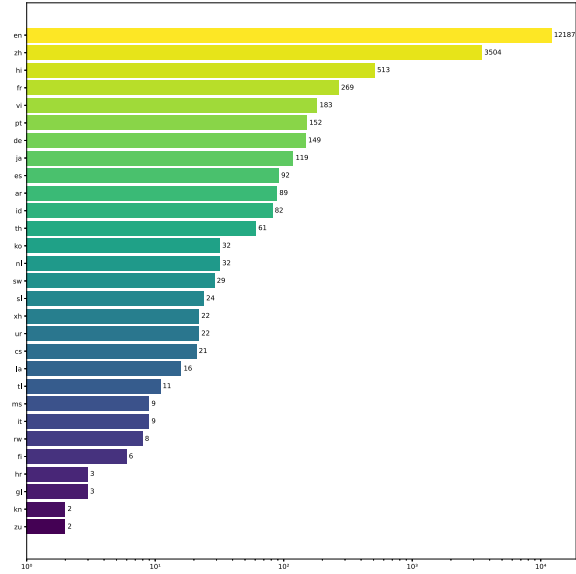


Figure 3: Language distribution of all tasks (Logarithmic Scale).

Our dataset consists of 17,540 tasks, including 14,295 single-tool calling tasks and 3,245 multiple tools calling tasks. The language distribution of all tasks is shown in Figure 3. For single-tool calling tasks, we ensure coverage across all countries and categories from which the APIs are sourced. While a significant portion of the APIs are from the USA (including global APIs), resulting in a higher volume of English-language queries, we have intentionally generated more tasks for long-tail APIs to mitigate the long-tail problem. As a result, global

APIs comprise 56.27% of the tasks, while long-tail APIs account for 13.57%. In the case of multiple tools calling tasks, each task typically requires the LLM to invoke between 2 and 5 tools to successfully complete the task.

## 5 Experiments and Results

### 5.1 Implementation Details

We used both open-source and closed-source LLMs in our experiments. Open-source models, which are freely accessible for research and development, include those with relatively strong general capabilities (such as Qwen2.5 (Yang et al., 2024) and Deepseek-V3 (Liu et al., 2024a)), as well as models dedicated to tool calling (such as Hammer2 (Lin et al., 2024b) and Watt-tool-8B). In contrast, closed-source models comprise advanced systems such as GPT-4o and Claude-3.5-Sonnet.

We applied the default parameters for the open-source LLMs during testing on our dataset. To fine-tune the models, we used LoRA (Hu et al., 2021), training them for 3 epochs with a batch size of 1 per device, 8 gradient accumulation steps, and a learning rate of  $1.0e-4$ . A cosine learning rate scheduler with a warmup ratio of 0.1 was used for the training configuration.

### 5.2 Evaluation Metrics

In this paper, we employ four evaluation metrics to comprehensively assess model performance. The first three—drawn from Seal-Tools (Wu et al., 2024)—are as follows: **(1) Tool Selection (P/R/F1)** evaluates the model’s ability to accurately identify the appropriate tools from a set of candidates, measuring tool localization accuracy via precision, recall, and F1-score; **(2) Tool Invocation (P/R/F1)** assesses the model’s capability to generate correct and complete invocation parameters, using triple-level matching to compute precision, recall, and F1; **(3) Format Matching Accuracy (FM)** measures whether the model’s output adheres to the required JSON schema, a critical prerequisite for successful downstream execution.

However, these metrics overlook a key requirement in multilingual real-world applications: the ability of large language models to maintain linguistic consistency throughout tool calling interactions. To address this, we introduce a new metric—**Language Matching Accuracy (LM)**—which quantifies the proportion of cases where the model’s internal reasoning (thought field)

is expressed in the same language as the user’s query. Language identification is performed using the langid library. Detailed computation methods for all four metrics are provided in Appendix C.

Model Name	LM	FM	Tool Selection			Tool Invocation		
			P	R	F1	P	R	F1
Qwen2.5-7B-Instruct	90.51	96.65	54.08	53.06	53.18	42.76	43.37	42.71
Qwen2.5-Coder-7B	94.93	98.38	69.76	66.01	67.23	54.17	54.11	53.75
Qwen2.5-3B-Instruct	87.40	93.00	49.34	45.84	47.52	40.90	41.77	41.33
Qwen2.5-Coder-3B	84.26	89.25	48.92	49.01	48.76	38.49	38.83	38.43
watt-tool-8B	74.48	5.53	<b>88.90</b>	<b>88.03</b>	<b>88.30</b>	<b>76.33</b>	<b>73.46</b>	<b>74.31</b>
ToolACE-8B	81.31	4.56	70.30	69.82	69.93	59.39	56.22	57.17
Hammer2.1-7b	86.82	20.71	64.64	64.68	64.44	33.14	32.68	32.75
Hammer2.0-7b	78.21	95.42	61.22	57.48	58.68	45.00	45.25	44.85
Functionary-v3.1	76.75	54.15	40.63	37.15	38.30	35.25	35.64	35.02
Yi-1.5-9B-Chat-16K	82.37	91.9	45.28	45.71	45.32	35.67	35.66	35.33
glm-4-9b-chat	76.00	97.55	43.45	41.44	42.09	32.77	32.85	32.57
Phi-4	<b>96.73</b>	96.29	80.90	82.68	81.49	70.15	70.25	69.84
Qwen2.5-Coder-32B	91.05	99.14	84.82	81.44	82.54	71.13	71.04	70.69
Qwen2.5-72B-Instruct	89.47	98.16	52.78	51.44	51.83	43.11	43.35	42.89
Deepseek-V3	86.09	99.89	83.10	83.73	83.28	75.94	75.77	75.49
Deepseek-R1	77.05	<b>100</b>	86.89	85.25	85.79	73.47	73.15	72.79
o1-mini	95.89	93.68	64.41	66.61	64.72	60.58	62.53	61.06
o3-mini	86.19	71.37	61.06	61.13	60.93	54.01	53.56	53.54
GPT4o-mini	96.24	<b>99.83</b>	76.47	75.21	75.55	71.69	70.38	70.71
GPT4o	<b>97.95</b>	99.83	<b>88.95</b>	<b>89.48</b>	<b>89.01</b>	<b>82.18</b>	<b>81.57</b>	<b>81.57</b>
GLM-Zero	88.37	98.45	51.24	50.31	50.51	42.64	43.64	42.78
gemini-2.0-flash	95.04	99.77	77.25	77.76	77.32	69.08	68.14	68.18
gemini-2.0-pro	96.17	94.13	84.57	83.50	83.86	73.22	71.65	71.95
Claude-3.5-sonnet	94.75	97.06	82.08	81.00	81.19	72.05	72.29	71.77

Table 2: Zero-shot evaluation results on ITC testing data (%). The best results are highlighted in bold.

### 5.3 Zero-Shot Evaluation of Tool Calling Capabilities

We begin by evaluating the zero-shot performance of large language models (LLMs) on our ITC testing dataset, aiming to assess their intrinsic tool calling capabilities without task-specific fine-tuning.

**Overall performance:** Table 2 presents a comprehensive comparison of both open-source and closed-source LLMs. As expected, closed-source models generally outperform open models across all evaluation dimensions. However, several open-source models still demonstrate strong performance on specific sub-tasks.

**Linguistic and structural accuracy:** In terms of **LM**, which evaluates the consistency and fluency of generated instructions, scores range from 74% to 97.95%. Open-source models such as Qwen2.5-7B-Instruct achieve high linguistic fidelity (90.51%), while closed models like GPT-4o reach the upper limit (97.95%). For **FM**—which measures adherence to the required output structure (e.g., JSON)—nearly all closed-source models (except o3-mini) and over two-thirds of open-source models achieve scores above 90%. Notably, Deepseek-R1 (100%) and Qwen2.5-Coder-32B (99.14%) exhibit excellent structural compliance, whereas models like watt-tool-8B and ToolACE-8B often produce malformed outputs.

**Functional competence in tool calling:** For **Tool Selection**, closed-source models such as GPT-4o exhibit superior performance, achieving high precision, recall, and F1 scores—demonstrating a strong understanding of task-to-tool alignment. Among open-source models, watt-tool-8B performs comparatively well, while others like Functionary-v3.1 frequently fail to identify appropriate tools, indicating weaknesses in tool relevance comprehension. In terms of **Tool Invocation**, GPT-4o and watt-tool-8B again lead with high scores, reflecting robust capabilities in generating accurate invocation parameters and producing well-structured outputs. In contrast, models like Hammer2.1-7B struggle significantly, often generating malformed invocations or incorrect arguments, highlighting deficiencies in parameter composition and structural formatting.

**Discrepancy analysis:** The performance gap between Tool Selection and Tool Invocation provides insight into a model’s holistic understanding of tool-use tasks. GPT-4o demonstrates high consistency across both stages, indicating mature reasoning and generation abilities. Conversely, models with a wider performance gap—such as Hammer2.1-7B—typically exhibit difficulties in translating tool choice into correct parameterization and invocation syntax. This divergence serves as a critical diagnostic signal for tool-augmented applications, reflecting weaknesses in planning, schema grounding, or structured generation.

**Error analysis:** Table 3 summarizes tool selection and invocation errors across models. Most models struggle with missing required tools and hallucinating non-existent ones. For example, gemini-2.0-pro avoids hallucination entirely (0%) but misses many tools (85.64%), while GPT4o achieves a more balanced performance. In invocation, incorrect and missing parameters are common issues, with Hammer2.1-7b showing high error rates. Models like GPT4o-mini and ToolACE-8B demonstrate more consistent performance across error types, suggesting better robustness in tool calling.

#### 5.4 Fine-tuned Evaluation of Tool Calling Capabilities

In this experiment, we fine-tuned four models from the Qwen 2.5 family—Qwen2.5-7B-Instruct, Qwen2.5-Coder-7B, Qwen2.5-3B-Instruct, and Qwen2.5-Coder-3B—on our ITC training dataset to evaluate the impact of fine-tuning on the tool calling capabilities of open-source LLMs.

**Testing results on the ITC test set:** Table 4 demon-

Model Name	Tool Selection			Tool Invocation		
	Hall.	Mis.	Ex.	Incor.	Miss.	Ext.
Qwen2.5-7B-Instruct	21.57	73.23	5.20	51.53	19.73	28.74
Qwen2.5-Coder-7B	4.25	86.65	9.10	51.01	20.59	28.39
Qwen2.5-3B-Instruct	8.74	75.79	15.47	42.14	<b>16.42</b>	41.45
Qwen2.5-Coder-3B	38.48	<b>51.51</b>	10.00	37.36	23.02	39.62
watt-tool-8B	25.51	67.74	6.74	45.54	40.61	13.85
ToolACE-8B	4.12	88.75	7.13	42.03	48.63	<b>9.34</b>
Hammer2.1-7b	0.70	91.56	7.74	<b>17.18</b>	64.26	18.56
Hammer2.0-7b	2.35	89.16	8.49	57.80	23.92	18.28
Functionary-v3.1	20.92	76.97	<b>2.11</b>	37.70	28.80	33.51
Yi-1.5-9B-Chat-16K	37.35	55.78	6.86	38.90	18.66	42.44
glm-4-9b-chat	0.98	93.90	5.12	27.88	54.78	17.34
Phi-4	11.29	69.18	19.53	44.75	29.11	26.14
Qwen2.5-Coder-32B	8.09	66.31	25.61	46.43	27.80	25.78
Qwen2.5-72B-Instruct	43.64	51.95	4.41	46.65	22.79	30.56
Deepseek-V3	<b>0.43</b>	80.21	19.36	57.65	24.38	17.97
Deepseek-R1	8.33	83.33	8.33	41.67	29.17	29.17
o1-mini	35.5	61.68	<b>2.82</b>	54.5	28.5	17.0
o3-mini	34.8	62.33	2.87	53.89	29.64	16.47
GPT4o-mini	19.71	76.26	4.03	48.76	39.94	<b>11.29</b>
GPT4o	47.16	<b>49.72</b>	3.12	53.67	21.22	25.10
GLM-Zero	37.63	58.09	4.28	48.89	<b>13.27</b>	37.83
gemini-2.0-flash	30.93	62.13	6.95	50.42	29.50	20.08
gemini-2.0-pro	<b>0</b>	85.64	14.36	<b>47.31</b>	35.13	17.56
Claude-3.5-sonnet	22.11	68.81	9.08	54.67	21.21	24.12

Table 3: Error analysis of tool selection and invocation across different LLMs (%). Hall.: hallucinating non-existing tools, Mis.: missing required tools, Ex.: calling extra tools, Incor.: generating incorrect parameters, Miss.: missing parameters, Ext.: generating extra parameters. The best results are highlighted in bold.

strates significant improvements in both tool selection and invocation following fine-tuning. Remarkably, the fine-tuned 3B models achieve performance on par with the larger 7B models. For example, Qwen2.5-7B-Instruct achieved a 45.02% increase in tool selection recall and a 47.88% increase in tool invocation precision, while Qwen2.5-Coder-3B recorded the highest improvement in tool invocation F1 score at 51.53%, with overall performance closely matching that of Qwen2.5-7B-Instruct. These findings confirm the robustness of our training data in boosting tool calling capabilities across different model scales.

**Testing results on out-of-domain data:** To evaluate the models’ ability to generalize beyond the training distribution, we tested our fine-tuned Qwen2.5 variants on challenging out-of-domain tool selection and invocation tasks. As shown in Table 5, all four models demonstrate significant improvements, with tool selection precision increasing by up to 25.75% and tool invocation precision improving by as much as 18.10%. These results highlight the effectiveness of our training dataset in enhancing performance on diverse, previously unseen tool calling tasks.

#### 5.5 Ablation Study on Language Impact

To assess the impact of non-English languages on model performance, we conducted an ablation

Model Name	LM	FM	Tool Selection			Tool Invocation		
			P	R	F1	P	R	F1
Qwen2.5-7B-Instruct	96.89(+6.38)	99.77(+3.12)	<b>97.72</b> (+43.64)	<b>98.08</b> (+45.02)	<b>97.76</b> (+44.58)	<b>90.64</b> (+47.88)	<b>90.55</b> (+47.18)	<b>90.34</b> (+47.63)
Qwen2.5-Coder-7B	<b>97.41</b> (+2.48)	99.64(+1.26)	97.69(+27.93)	98.00(+31.99)	97.72(+30.49)	90.57(+36.4)	90.38(+36.27)	90.22(+36.47)
Qwen2.5-3B-Instruct	97.26(+9.86)	99.54(+6.54)	97.35(+48.01)	97.92(+ <b>52.08</b> )	97.48(+ <b>49.96</b> )	89.78(+ <b>48.88</b> )	89.50(+47.73)	89.36(+48.03)
Qwen2.5-Coder-3B	97.29( <b>+13.03</b> )	<b>99.79</b> ( <b>+10.54</b> )	97.64( <b>+48.72</b> )	97.89(+48.88)	97.64(+48.88)	90.25(+51.76)	90.26(+ <b>51.43</b> )	89.96(+ <b>51.53</b> )

Table 4: Fine-tuned evaluation results on ITC testing dataset (%), with values in brackets showing the improvement from the original models. The best results and greatest improvements are highlighted in bold.

Model	Nexus Raven		Seal-Tools		Tool-Alpaca	
	Tool Selection (P)	Tool Invocation (P)	Tool Selection (P)	Tool Invocation (P)	Tool Selection (P)	Tool Invocation (P)
Qwen2.5-7B-Instruct	90.57(+ <b>25.75</b> )	59.97(+10.23)	89.91(+ <b>24.00</b> )	76.16(+17.76)	<b>77.05</b> (+ <b>18.10</b> )	49.96(+ <b>9.85</b> )
Qwen2.5-Coder-7B-Instruct	<b>90.99</b> (+20.44)	<b>68.04</b> (+ <b>17.76</b> )	89.57(+22.22)	<b>78.04</b> (+18.92)	77.34(+14.91)	<b>50.87</b> (+8.23)
Qwen2.5-3B-Instruct	81.03(+6.50)	57.14(+2.78)	<b>90.32</b> (+23.26)	76.76(+ <b>20.02</b> )	75.00(+8.92)	47.54(+8.57)
Qwen2.5-Coder-3B-Instruct	84.17(+2.09)	64.22(+4.90)	89.34(+8.50)	76.18(+7.97)	73.10(+4.53)	48.69(+7.06)

Table 5: Fine-tuned evaluation results on three benchmark testing datasets (%), with values in brackets showing the improvement from the original models. The best results and greatest improvements are highlighted in bold.

Type	Model Name	LM	FM	Tool Selection			Tool Invocation		
				P	R	F1	P	R	F1
ALL	Qwen2.5-7B-Instruct	96.30(+5.56)	99.27(+4.18)	91.57(+36.67)	<b>98.57</b> (+42.02)	<b>94.94</b> (+39.69)	87.78(+45.93)	86.36(+44.08)	87.06(+45.45)
	Qwen2.5-Coder-7B	<b>96.47</b> (+7.04)	<b>98.91</b> (+1.46)	<b>93.29</b> (+21.77)	93.55(+19.61)	93.42(+20.67)	<b>88.37</b> (+33.17)	<b>89.19</b> (+34.80)	<b>88.77</b> (+34.56)
	Qwen2.5-3B-Instruct	91.62(+ <b>11.24</b> )	95.91(+ <b>13.55</b> )	87.03(+39.58)	89.76(+40.25)	88.37(+ <b>39.91</b> )	76.17(+44.06)	74.23(+41.27)	75.19(+42.66)
	Qwen2.5-Coder-3B	94.21(+9.34)	<b>98.91</b> (+4.36)	87.44(+ <b>40.15</b> )	86.91(+39.30)	87.17(+39.86)	80.20(+ <b>46.89</b> )	80.42(+ <b>45.66</b> )	80.31(+ <b>46.67</b> )
EN	Qwen2.5-7B-Instruct	91.33(+0.59)	97.09(+2.00)	79.55(+24.65)	79.12(+22.57)	79.33(+24.08)	70.82(+28.97)	71.17(+28.89)	70.99(+29.38)
	Qwen2.5-Coder-7B	<b>92.57</b> (+2.14)	<b>98.28</b> (+1.17)	<b>88.24</b> (+15.72)	<b>88.32</b> (+14.38)	<b>88.28</b> (+15.53)	<b>79.47</b> (+24.27)	<b>79.63</b> (+25.24)	<b>79.55</b> (+25.34)
	Qwen2.5-3B-Instruct	83.64(+ <b>3.26</b> )	86.51(+ <b>4.15</b> )	77.96(+30.51)	77.70(+28.19)	77.83(+29.37)	69.7(+ <b>37.59</b> )	69.97(+ <b>37.01</b> )	69.83(+ <b>37.30</b> )
	Qwen2.5-Coder-3B	85.67(+0.80)	96.45(+1.90)	78.85(+ <b>31.56</b> )	79.47(+ <b>31.86</b> )	79.16(+ <b>31.85</b> )	69.85(+36.54)	69.99(+35.23)	69.92(+36.28)

Table 6: Ablation study on non-English queries in the ITC testing dataset evaluating language impact (%), with values in brackets indicating improvements over the original models. The label ‘Type = ALL’ denotes training on the full ITC dataset, while ‘Type = EN’ indicates training exclusively on the English subset of the ITC dataset. The best results and largest improvements are highlighted in bold.

study by fine-tuning Qwen2.5 models either on the full multilingual ITC training set (‘Type = ALL’) or exclusively on the English subset (‘Type = EN’), then evaluating on the non-English test data. As shown in Table 6, models trained on the full dataset achieve substantially higher gains on non-English queries. For example, the Qwen2.5-7B-Instruct model fine-tuned on all languages improved tool selection recall by 42.02%, which is 19.45% higher than the gain achieved by the same model trained only on English. Similarly, tool invocation F1 for Qwen2.5-Coder-7B increased by 34.56% with full multilingual training, outperforming the English-only training gain by 9.22%. These results demonstrate that limiting training to English significantly restricts performance gains on non-English queries, highlighting the importance of incorporating diverse language data to strengthen the multilingual generalization capabilities of large language models.

## 6 Conclusion

In this paper, we address the urgent need for a diverse, globally representative dataset to advance large language models’ tool calling capabilities. We introduce the International Tool Calling (ITC) dataset—a comprehensive resource designed to train and evaluate LLMs across international and multi-tool calling scenarios. Covering a broad spectrum of API categories, including both global and region-specific APIs, ITC effectively addresses challenges such as long-tail API coverage and the complexities of multi-tool interactions. Our experiments highlight the dataset’s value in uncovering critical performance issues in tool invocation, including handling missing or incorrect parameters, and demonstrate substantial improvements achievable through fine-tuning. These results underscore ITC’s potential to enhance LLMs’ proficiency with international APIs and point toward promising avenues for future research.



## Limitations

While our work presents significant advancements, several limitations warrant further attention. First, despite emphasizing geographical diversity, certain regions (e.g., Africa and parts of Asia) remain underrepresented, potentially limiting the model’s ability to grasp nuanced cultural or regulatory contexts. Second, the dataset focuses solely on REST APIs, leaving other tool types (e.g., SOAP APIs or database connectors) unaddressed, which may constrain applicability in more heterogeneous tool ecosystems. Third, reliance on free APIs introduces potential instability due to service deprecation or rate limits, making regular dataset updates essential to maintain relevance and reproducibility. Finally, more challenging datasets are needed to further boost the tool calling capabilities of open-source LLMs. Addressing these issues will be critical for future work aimed at building truly robust and universal tool calling systems.

## Acknowledgments

## References

Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis A. Lastras. 2024. [Api-blend: A comprehensive corpora for training and benchmarking api llms](#). *Preprint*, arXiv:2402.15491.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

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

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Tenghao Huang, Dongwon Jung, and Muhao Chen. 2024. Planning and editing what you retrieve for enhanced tool learning. *arXiv preprint arXiv:2404.00450*.

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. 2024a. [Hammer: Robust function-calling for on-device language models via function masking](#). *Preprint*, arXiv:2410.04587.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. 2024b. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.

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](#). *Preprint*, arXiv:2409.00920.

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. 2024c. [Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets](#). *ArXiv*, abs/2406.18518.

Marco Lui and Timothy Baldwin. 2012. [langid.py: An off-the-shelf language identification tool](#). In *Proceedings of the ACL 2012 System Demonstrations*, pages 25–30, Jeju Island, Korea. Association for Computational Linguistics.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, Aixin Sun, Hany Awadalla, et al. 2024. Sciaagent: Tool-augmented language models for scientific reasoning. *arXiv preprint arXiv:2402.11451*.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. [Talm: Tool augmented language models](#). *Preprint*, arXiv:2205.12255.

729	Shishir G. Patil, Tianjun Zhang, Xin Wang, and	Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang,	785
730	Joseph E. Gonzalez. 2023. <a href="#">Gorilla: Large lan-</a>	Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou,	786
731	<a href="#">guage model connected with massive apis</a> . <i>Preprint</i> ,	Qi Zhang, Tao Gui, et al. 2024a. Tooleyes: Fine-	787
732	arXiv:2305.15334.	grained evaluation for tool learning capabilities of	788
		large language models in real-world scenarios. <i>arXiv</i>	789
733	Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan	<i>preprint arXiv:2401.00741</i> .	790
734	Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang,		
735	Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian,	Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang,	791
736	Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li,	Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui,	792
737	Zhiyuan Liu, and Maosong Sun. 2023. <a href="#">Toolllm: Fa-</a>	and Xuanjing Huang. 2024b. Toolsword: Un-	793
738	<a href="#">cilitating large language models to master 16000+</a>	veiling safety issues of large language models in	794
739	<a href="#">real-world apis</a> . <i>Preprint</i> , arXiv:2307.16789.	tool learning across three stages. <i>arXiv preprint</i>	795
		<i>arXiv:2402.10753</i> .	796
740	Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta		
741	Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola	Junjie Ye, Yilong Wu, Songyang Gao, Caishuang	797
742	Cancedda, and Thomas Scialom. 2023. <a href="#">Toolformer:</a>	Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang,	798
743	<a href="#">Language models can teach themselves to use tools</a> .	Tao Gui, and Xuanjing Huang. 2024c. Rotbench: a	799
744	<i>Preprint</i> , arXiv:2302.04761.	multi-level benchmark for evaluating the robustness	800
		of large language models in tool learning. <i>arXiv</i>	801
745	Simranjit Singh, Michael Fore, and Dimitrios Stamoulis.	<i>preprint arXiv:2401.08326</i> .	802
746	2024. Evaluating tool-augmented agents in remote		
747	sensing platforms. <i>arXiv preprint arXiv:2405.00709</i> .	Alex Young, Bei Chen, Chao Li, Chengen Huang,	803
		Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng	804
748	Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei	Li, Jiangcheng Zhu, Jianqun Chen, et al. 2024. Yi:	805
749	Han, Qiao Liang, Boxi Cao, and Le Sun. 2023a.	Open foundation models by 01. ai. <i>arXiv preprint</i>	806
750	<a href="#">Toolalpaca: Generalized tool learning for language</a>	<i>arXiv:2403.04652</i> .	807
751	<a href="#">models with 3000 simulated cases</a> . <i>Preprint</i> ,		
752	arXiv:2306.05301.	Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan	808
		Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin,	809
753	Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei	Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023.	810
754	Han, Qiao Liang, Boxi Cao, and Le Sun. 2023b.	Judging llm-as-a-judge with mt-bench and chatbot	811
755	Toolalpaca: Generalized tool learning for language	arena. <i>Advances in Neural Information Processing</i>	812
756	models with 3000 simulated cases. <i>arXiv preprint</i>	<i>Systems</i> , 36:46595–46623.	813
757	<i>arXiv:2306.05301</i> .		
758	Gemini Team, Rohan Anil, Sebastian Borgeaud,		
759	Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu,		
760	Radu Soricut, Johan Schalkwyk, Andrew M Dai,		
761	Anja Hauth, et al. 2023. Gemini: a family of		
762	highly capable multimodal models. <i>arXiv preprint</i>		
763	<i>arXiv:2312.11805</i> .		
764	Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan		
765	Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer,		
766	Damien Vincent, Zhufeng Pan, Shibo Wang, et al.		
767	2024. Gemini 1.5: Unlocking multimodal under-		
768	standing across millions of tokens of context. <i>arXiv</i>		
769	<i>preprint arXiv:2403.05530</i> .		
770	Chenyu Wang, Weixin Luo, Qianyu Chen, Haonan		
771	Mai, Jindi Guo, Sixun Dong, Zhengxin Li, Lin Ma,		
772	Shenghua Gao, et al. 2024. Tool-llm: A large multi-		
773	modal model for tool agent learning. <i>arXiv preprint</i>		
774	<i>arXiv:2401.10727</i> .		
775	Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan,		
776	Xiang Zhang, and Wenliang Chen. 2024. Seal-tools:		
777	self-instruct tool learning dataset for agent tuning and		
778	detailed benchmark. In <i>CCF International Confer-</i>		
779	<i>ence on Natural Language Processing and Chinese</i>		
780	<i>Computing</i> , pages 372–384. Springer.		
781	An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui,		
782	Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu,		
783	Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 tech-		
784	nical report. <i>arXiv preprint arXiv:2412.15115</i> .		

A API Processing  
A.1 API Format



Figure 4: API Format.

A.3 API Empty Response

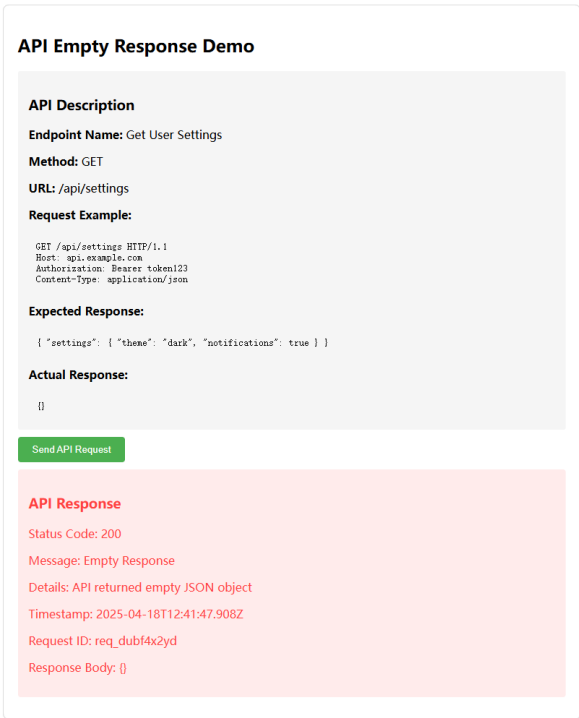


Figure 6: API Empty Response Demo.

A.2 API Error Response

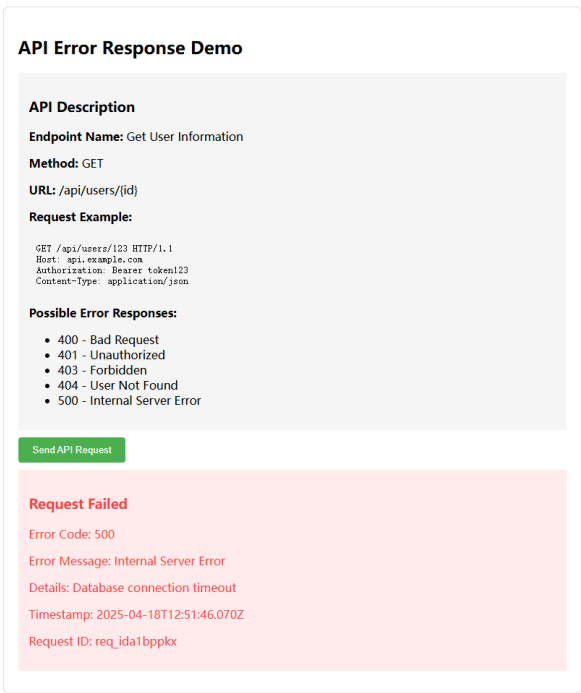


Figure 5: API Error Response Demo.

A.4 API Documentation Rewriting

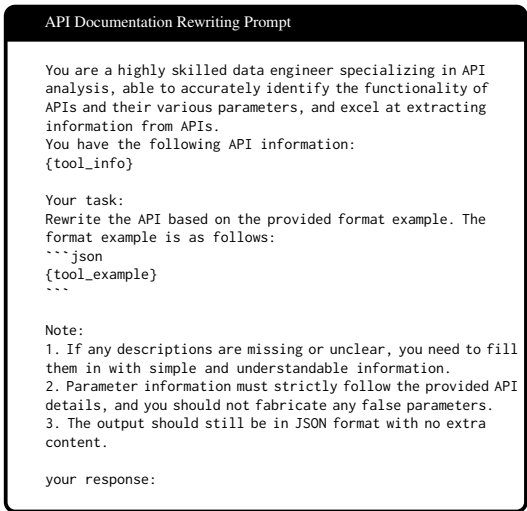


Figure 7: API Documentation Rewriting.

A.5 API Country Distribution

Figure 8 provides a comprehensive overview of the geographical distribution of APIs in our dataset, including both global and country-specific APIs across more than 30 countries and regions.

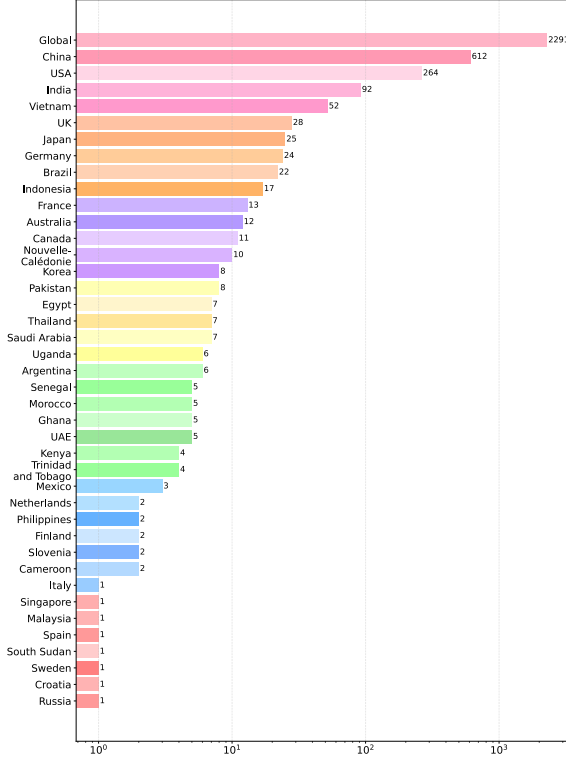


Figure 8: Distribution of tools by countries (Logarithmic Scale).

## B Annotation Protocol and Details

All annotations in this study were performed by the authors themselves.

### B.1 Instructions

The authors followed a predefined set of evaluation guidelines when rating the generated instructions. Each item was scored along five dimensions: accuracy, fluency, cultural adaptability, grammatical correctness, and feasibility of execution. Each dimension was rated on a scale from 1 (poor) to 5 (excellent). The annotators discussed unclear cases and followed the same rubric throughout the evaluation process. The full text of the annotation instructions is shown in Figure 14

### B.2 Recruitment and Payment

No external annotators were recruited. All annotation work was carried out by the authors as part of the research effort. Therefore, no monetary compensation was involved.

### B.3 Annotator Characteristics

The annotators are the co-authors of this paper. All are NLP researchers with prior experience in multi-lingual evaluation and large language model devel-

opment. They are fluent in the relevant languages used in the dataset. This ensured high-quality and consistent annotation across multiple languages and tasks.

## C Detailed Formulate for Evaluation Metrics

To control page layout, we use **FM** to represent *Format Matching Accuracy*, **LM** for *Language Matching Accuracy*, **Tool** for *Tool Selection*, and **TI** for *Tool Invocation*.

$$LM = \frac{amount_{correct\ language}}{amount_{all}} \quad (1)$$

$$FM = \frac{amount_{correct\ format}}{amount_{all}} \quad (2)$$

$$Tool\ P = \frac{amount_{correct\ tools}}{amount_{predict\ tools}} \quad (3)$$

$$Tool\ R = \frac{amount_{correct\ tools}}{amount_{gold\ tools}} \quad (4)$$

$$Tool\ F1 = \frac{2 \cdot Tool\ P \cdot Tool\ R}{Tool\ P + Tool\ R} \quad (5)$$

$$TI\ P = \frac{amount_{correct\ tools}}{amount_{predict\ tools}} \quad (6)$$

$$TI\ R = \frac{amount_{correct\ tools}}{amount_{gold\ tools}} \quad (7)$$

$$TI\ F1 = \frac{2 \cdot TI\ P \cdot TI\ R}{TI\ P + TI\ R} \quad (8)$$

## D Single Tool Calling Tasks Query Generation Prompt

For single tool calling tasks, we utilize a prompt-based approach to instruct the LLM to generate a query. The prompt templates used for this process are illustrated in Figures 9.

## E Multiple tools Calling Tasks Query Generation

For multiple tool calling tasks, we have classified them into three categories: Repeated Calls, Parallel Calls, and Nested Calls. Given that the requirements for each type of task differ, we have tailored specific prompts to generate queries for each category. The prompt templates for these tasks are illustrated in Figures 10, 11, and 12.



### Single Tool Calling Tasks Query Generation Prompt

Please strictly follow these guidelines:

1. The instructions should be 1 to 2 sentences long. Use a mix of interrogative sentences, first-person statements, imperative sentences, and other structures that convey a request. Aim for diversity in your instructions.
2. Do not mention the API's name in your instructions.
3. Your instructions should only involve the features provided by these APIs.
4. Generate 10 diverse instructions.
5. Use specific nouns and real-world examples from various domains, such as entertainment, sports, or technology.
6. Please provide concrete details. Don't using any form of generic phrases, such as "this xxx", "the xxx", "a xxx" or "a specific xxx".
7. Ensure diversity in language by combining questions with imperative statements and other structures that convey a request.
8. The instructions should be in the language of the country attribute in the provided API information.
9. The generated problem must strictly follow the API's parameter information.
10. If country is Global, please generate 10 instructions in English.

Here is the API information:

{api\_list}

Please generate the question in the language of the specified country.

your response:

Figure 9: Query generation prompt for single tool calling tasks.

## F Query Scoring

### F.1 Scoring dimensions

To comprehensively assess the quality of instructions (queries or question-and-answer pairs), we adopt the following five evaluation dimensions:

1. **Relevance:** Measures the alignment between the instruction and the task objective. High-scoring instructions accurately reflect the task requirements, while irrelevant or off-topic instructions receive lower scores.
2. **Practicality:** Assesses the feasibility and executability of the instruction in real-world scenarios. High scores indicate instructions that can be directly implemented without significant obstacles.
3. **Linguistic Applicability:** Evaluates the instruction's adherence to grammatical norms and consideration of cultural and linguistic context. High-scoring instructions are well-phrased, natural, and unambiguous.
4. **Clarity:** Judges whether the instruction is clearly articulated, logically coherent, and easy to understand. High scores indicate concise, explicit, and actionable instructions.
5. **Specificity:** Measures the level of detail and focus in the instruction. High-scoring instruc-

tions clearly define the scope of operation, reduce ambiguity, and facilitate precise tool invocation.

Each dimension is scored on a scale from 1 to 5, where 1 indicates very low quality and 5 indicates very high quality. The detailed scoring criteria are shown in Table 7

### F.2 Example of scoring

Figure 13 illustrates an example of query scoring, where, given a query and relevant API information, we used both Anthropic's Claude-3.5-sonnet model and Google's Gemini-1.5-pro model to evaluate the query's quality across five dimensions, with scores ranging from 1 to 5 for each dimension. Figure 14 shows the prompt for LLMs to evaluate the query.

Dimension	1 (Very Low)	3 (Medium)	5 (Very High)
Relevance	Completely irrelevant	Partially relevant	Highly aligned with the task
Practicality	Hard to implement	Feasible but with obstacles	Directly applicable
Linguistic Applicability	Incorrect or awkward	Basically fluent	Standard, natural, idiomatic
Clarity	Vague or confusing	Understandable but ambiguous	Clear and unambiguous
Specificity	Overly broad or vague	Some details present	Detailed, focused, unambiguous

Table 7: Scoring guidelines for each evaluation dimension.

Multiple tool Repeated Calling Tasks Query Generation

Please strictly follow these guidelines:

1. Provide 1 to 2 sentences for each instruction, using a mix of interrogative sentences, imperative statements, and descriptive requests. Ensure the instructions are diverse in structure and tone to reflect real-world scenarios.
2. For each instruction, select only one tool (from the provided API list) and perform multiple calls to the same tool to complete different tasks.
3. Do not mention any API names directly in your instructions. Instead, focus on the functionality they provide.
4. Leverage the unique features of the selected tool. Each instruction must clearly demonstrate how the tool can be used through multiple calls to effectively solve a problem or fulfill a request. Avoid generic or vague task descriptions.
5. Use detailed and specific nouns, examples, and contextual scenarios from domains like travel, business, sports, entertainment, or technology. Avoid vague phrases such as "this information" or "a specific detail."
6. The generated instructions must strictly align with the parameter information of each API/tool. Ensure the inputs and outputs are valid for the respective tools.
7. Generate 10 diverse instructions, each showcasing a single tool being used multiple times. Each instruction can use a different tool.
8. Ensure the content of each instruction strictly aligns with the examples provided and closely follows the format of the examples below.

Here are some examples for Multi-Tool Instructions:  
{example\_list}

Here is the API information:  
{api\_data}

Here is the Output format:  
{Output\_format}

Please select only one tool (from the provided API list) and strictly following the Output format.

your response:

Figure 10: Multiple tool repeated calls.

G Data Examples

Figure 15 illustrates an example of the Google Translate API. Figure 16 provides an example of a single tool calling task, while Figure 17 demonstrates a repeated multiple tools calling task. Figure 18 shows an example of a parallel multiple tools calling task, and Figure 19 presents an example of a nested multiple tools calling task.

Multiple tool Parallel Calling Tasks Query Generation

Please strictly follow these guidelines:

1. Provide 1 to 2 sentences for each instruction, using a mix of interrogative sentences, imperative statements, and descriptive requests. Ensure the instructions are diverse in structure and tone to reflect real-world scenarios.
2. For each instruction, select only one tool (from the provided API list) and perform multiple calls to the same tool to complete different tasks.
3. Do not mention any API names directly in your instructions. Instead, focus on the functionality they provide.
4. Leverage the unique features of the selected tool. Each instruction must clearly demonstrate how the tool can be used through multiple calls to effectively solve a problem or fulfill a request. Avoid generic or vague task descriptions.
5. Use detailed and specific nouns, examples, and contextual scenarios from domains like travel, business, sports, entertainment, or technology. Avoid vague phrases such as "this information" or "a specific detail."
6. The generated instructions must strictly align with the parameter information of each API/tool. Ensure the inputs and outputs are valid for the respective tools.
7. Generate 10 diverse instructions, each showcasing a single tool being used multiple times. Each instruction can use a different tool.
8. Ensure the content of each instruction strictly aligns with the examples provided and closely follows the format of the examples below.

Here are some examples for Multi-Tool Instructions:  
{example\_list}

Here is the API information:  
{api\_data}

Here is the Output format:  
{Output\_format}

Please select only one tool (from the provided API list) and strictly following the Output format.

your response:

Figure 11: Multiple tool parallel calls.

Multiple tool Nested Calling Tasks Query Generation

Please strictly follow these guidelines:

1. Provide 1 to 2 sentences for each instruction, using a mix of interrogative sentences, imperative statements, and descriptive requests. Ensure the instructions are diverse in structure and tone to reflect real-world scenarios.
2. For each instruction, select exactly 2 or 3 tools from the provided API list to create a scenario where these tools are used in a logical sequence. Ensure that the output of the previous tool can serve as the input for the next tool, forming a nested call.
3. Do not mention any API names directly in your instructions. Instead, focus on the functionality they provide.
4. Use detailed and specific nouns, examples, and contextual scenarios from domains like travel, business, sports, entertainment, or technology. Avoid vague phrases such as "this information" or "a specific detail."
5. The generated instructions must strictly align with the parameter information of each API/tool. Ensure the inputs and outputs are valid for the respective tools.
6. Generate 10 diverse instructions, each involving 2 or 3 tools working together in a logical sequence and existence of nested calls.

Here are some examples for Multi-Tool Instructions:

{example\_list}

Here is the API information:

{api\_data}

Here is the Output format:

{Output\_format}

Please strictly following the Output format.

your response:

Figure 12: Multiple tool nested calls.

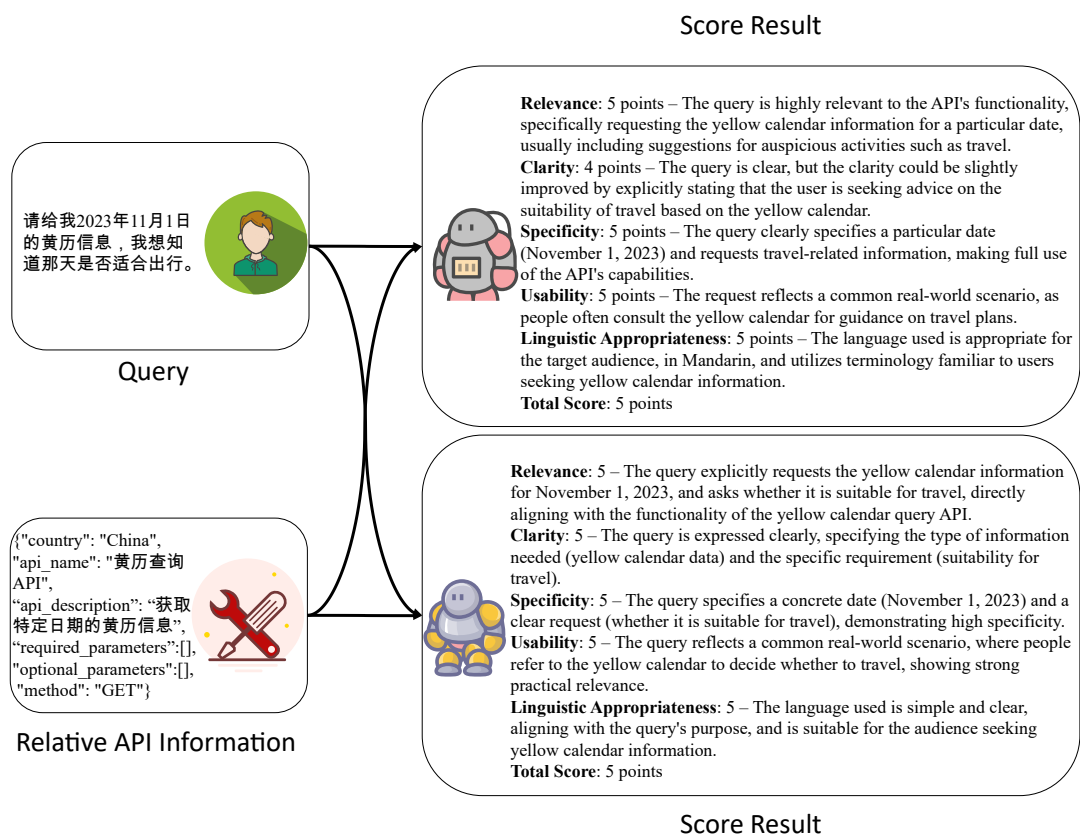


Figure 13: The query scoring process.



Query Scoring Prompt

Evaluation Criteria:

Use a 1-5 scale to score the following five dimensions:

1. Relevance: How well the query matches the API's functionality.
2. Clarity: Whether the query is specific enough, avoiding ambiguous terms like 'this xxx', 'the xxx', or 'a xxx', and ensuring the use of the API's features.
3. Specificity: Whether the query is specific enough to utilize the API's capabilities
4. Practicality: Whether the query reflects real-world usage scenarios
5. Language Appropriateness: Whether the query's language is suitable for target users

Scoring Standard:

- 1 point: Does not meet the standard
- 2 points: Partially meets the standard
- 3 points: Meets the basic standard
- 4 points: Meets the standard well
- 5 points: Fully meets the standard

Total Score Calculation:

Calculate the average of the five dimension scores, round to the nearest integer, as the final total score (1-5 points).

Evaluation Steps:

1. Carefully read the API name and the generated query.
2. Score each dimension and provide a brief explanation.
3. Calculate the total score.
4. Provide an overall evaluation and suggestions for improvement.
5. If the total score is less than 3, mark it as "Needs Improvement".

Output Format:

Scores:

1. Relevance: [Score] - [Explanation]
2. Clarity: [Score] - [Explanation]
3. Specificity: [Score] - [Explanation]
4. Practicality: [Score] - [Explanation]
5. Language Appropriateness: [Score] - [Explanation]

Total Score: [1-5 points]

Overall Evaluation:

[Brief summary of the query's strengths and weaknesses]

Improvement Suggestions:

[Provide specific suggestions for improvement if needed]

Conclusion: [If total score >= 4, then "Pass"; if total score < 4, then "Needs Improvement"]

Please evaluate the following data's query

```
{data['query']}
```

Your response:

Figure 14: Query scoring prompt.

API Example

```
{
  "tool_name": "Google Translate",
  "tool_description": "A tool for translating
text between different languages using
Google's translation services.",
  "home_url": "https://rapidapi.com/
nickrapidapi/api/google_translate/",
  "country": "Global",
  "api_list": [
    {
      "name": "translate1",
      "url": "https://google-translate.p.rapidapi.
com/",
      "description": "This API endpoint allows users
to perform translations of text from
one language to another.",
      "method": "GET",
      "category": "Utilities",
      "required_parameters": [
        {
          "name": "text",
          "type": "string",
          "description": "The text content to be
translated"
        },
        {
          "name": "target_lang",
          "type": "string",
          "description": "The target language code
(e.g., 'en' for English, 'zh' for Chinese)"
        }
      ],
      "optional_parameters": [],
      "statusCode": "200",
      "schema": {}
    }
  ]
}
```

Figure 15: Example of google translate API.

### Single Tool Calling Task Example

```
{"system": "You are an advanced AI assistant with expertise in:  
1. Intelligent tool and API selection  
2. Multilingual problem-solving  
3. Systematic analytical thinking  
Task Guidelines:  
    - Carefully analyze the problem presented  
    - Select the most appropriate tool(s) or API(s) to solve the task  
    - Provide clear, structured reasoning for your tool selection  
    - Ensure precise and efficient problem resolution  
  
Important Principles:  
    - If no suitable tool exists, clearly explain the limitations  
    - Prioritize accuracy and relevance in tool selection  
    - Consider potential alternative approaches  
    - Maintain a logical and methodical approach  
  
When tools are insufficient or parameters are incomplete:  
    - Explicitly state the gaps in available tools or information  
    - Provide constructive suggestions for obtaining resources  
    - Return an empty list in the \"Action\" section  
Note: Please respond in the same national language as the problem  
while maintaining clarity, logic, and structure in your answers.  
Below is the list of functions you can invoke (in JSON format):  
{api_list}  
  
Please strictly follow the format below, without additional text:  
  
json{\"Thought\": \"Respond in same language as problem. Provide  
analysis, reasoning for tool selection, and challenges.\",  
\"Action\": \"[function_name1(param_1='val1', param_2='val2'),  
function_name2(param_1='val1', param_2='val2')]\"},  
\"conversations\": [  
{  
  \"from\": \"user\",  
  \"value\": \"Can you help me translate 'Hello, how are you?' into French for my  
upcoming trip to Paris?\"  
},  
{  
  \"from\": \"assistant\",  
  \"value\": \"json{\"Thought\": \"To assist the user with their request for  
translation, I will utilize the Google Translate API, which is specifically  
designed for translating text between languages. The user needs a translation  
for a common greeting in French, which is straightforward.\",  
\"Action\": \"[Google Translate(text='Hello, how are you?', 'target_lang'='fr')]  
\"}\"}]\"}]}
```

Figure 16: Single tool calling task example.

#### Repeated Multiple Tools Calling Task Example

```
{
  "system": "You are an advanced AI assistant with expertise in:
  1. Intelligent tool and API selection
  2. Multilingual problem-solving
  3. Systematic analytical thinking
  Task Guidelines:
    - Carefully analyze the problem presented
    - Select the most appropriate tool(s) or API(s) to solve the task
    - Provide clear, structured reasoning for your tool selection
    - Ensure precise and efficient problem resolution

  Important Principles:
    - If no suitable tool exists, clearly explain the limitations
    - Prioritize accuracy and relevance in tool selection
    - Consider potential alternative approaches
    - Maintain a logical and methodical approach

  When tools are insufficient or parameters are incomplete:
    - Explicitly state the gaps in available tools or information
    - Provide constructive suggestions for obtaining resources
    - Return an empty list in the \"Action\" section

  Note: Please respond in the same national language as the problem
  while maintaining clarity, logic, and structure in your answers.
  Below is the list of functions you can invoke (in JSON format):
  {api_list}

  Please strictly follow the format below, without additional text:

  json{
    \"Thought\": \"Respond in same language as problem. Provide
    analysis, reasoning for tool selection, and challenges.\",
    \"Action\": \"[function_name1(param_1='val1', param_2='val2'),
    function_name2(param_1='val1', param_2='val2')]\"
  },
  \"conversations\": [
    {
      \"from\": \"user\",
      \"value\": \"Can you verify the email address 'john.doe@example.com' using the
      Express option? After that, check 'jane.smith@sample.com' with the Premium
      option to see if it's valid.\"
    },
    {
      \"from\": \"assistant\",
      \"value\": \"json{
        \"Thought\": \"To verify the email addresses, I will use the
        Global Email V4 API. First, I will check 'john.doe@example.com' with the Express
        option, and then I will verify 'jane.smith@sample.com' using the Premium option.
        \"Action\": \"[Global Email V4(opt='VerifyMailbox:Express', email=
        'john.doe@example.com'), Global Email V4(opt='VerifyMailbox:ExpressPremium',
        email='jane.smith@sample.com')]
        \"}\"
      }\"
    }
  ]
}
```

Figure 17: Repeated multiple tools calling task example.

#### Parallel Multiple Tools Calling Task Example

```
{
  "system": "You are an advanced AI assistant with expertise in:
  1. Intelligent tool and API selection
  2. Multilingual problem-solving
  3. Systematic analytical thinking
  Task Guidelines:
    - Carefully analyze the problem presented
    - Select the most appropriate tool(s) or API(s) to solve the task
    - Provide clear, structured reasoning for your tool selection
    - Ensure precise and efficient problem resolution

  Important Principles:
    - If no suitable tool exists, clearly explain the limitations
    - Prioritize accuracy and relevance in tool selection
    - Consider potential alternative approaches
    - Maintain a logical and methodical approach

  When tools are insufficient or parameters are incomplete:
    - Explicitly state the gaps in available tools or information
    - Provide constructive suggestions for obtaining resources
    - Return an empty list in the \"Action\" section
  Note: Please respond in the same national language as the problem
  while maintaining clarity, logic, and structure in your answers.
  Below is the list of functions you can invoke (in JSON format):
  {api_list}

  Please strictly follow the format below, without additional text:

  json{
    \"Thought\": \"Respond in same language as problem. Provide
    analysis, reasoning for tool selection, and challenges.\",
    \"Action\": \"[function_name1(param_1='val1', param_2='val2'),
    function_name2(param_1='val1', param_2='val2')]\"
  }\",
  \"conversations\": [
    {
      \"from\": \"user\",
      \"value\": \"Can you check the latest exchange rates for USD and AUD against EUR,
      and also give me a list of all available currencies?\"
    },
    {
      \"from\": \"assistant\",
      \"value\": \"json{
        \"Thought\": \"To fulfill the request, I need to retrieve the
        latest exchange rates for USD and AUD against EUR, as well as a list of all
        available currencies. This requires using two different API endpoints: one for
        the latest exchange rates and another for the list of currencies.\",
        \"Action\": \"[latest(from='EUR', to='USD,AUD'), currencies()]
        \"}\"
      }\"
    }
  ]
}
```

Figure 18: Parallel multiple tools calling task example.



### Nested Multiple Tools Calling Task Example

```
{
  "system": "You are an advanced AI assistant with expertise in:
  1. Intelligent tool and API selection
  2. Multilingual problem-solving
  3. Systematic analytical thinking
  Task Guidelines:
    - Carefully analyze the problem presented
    - Select the most appropriate tool(s) or API(s) to solve the task
    - Provide clear, structured reasoning for your tool selection
    - Ensure precise and efficient problem resolution

  Important Principles:
    - If no suitable tool exists, clearly explain the limitations
    - Prioritize accuracy and relevance in tool selection
    - Consider potential alternative approaches
    - Maintain a logical and methodical approach

  When tools are insufficient or parameters are incomplete:
    - Explicitly state the gaps in available tools or information
    - Provide constructive suggestions for obtaining resources
    - Return an empty list in the \"Action\" section

  Note: Please respond in the same national language as the problem
  while maintaining clarity, logic, and structure in your answers.
  Below is the list of functions you can invoke (in JSON format):
  {api_list}

  Please strictly follow the format below, without additional text:

  json{
    \"Thought\": \"Respond in same language as problem. Provide
    analysis, reasoning for tool selection, and challenges.\",
    \"Action\": \"[function_name1(param_1='val1', param_2='val2'),
    function_name2(param_1='val1', param_2='val2')]\",
    \"conversations\": [
      {
        \"from\": \"user\",
        \"value\": \"Can you generate a secure password with numbers and then hash that
        password using the MD5 algorithm?\"
      },
      {
        \"from\": \"assistant\",
        \"value\": \"json{
          \"Thought\": \"To generate a secure password with numbers, I will
          use the 'Generate password' API with the appropriate parameters. After
          generating the password, I will hash it using the 'MD5 Text Hash' API.\",
          \"Action\": \"[Generate password(numbers=true,pwCount=1,length=12),MD5 Text
          Hash(dataString='generated_password')]\",
          \"\"}
        },
        {
          \"role\": \"Tool\",
          \"content\": {
            \"code\": 200,
            \"msg\": \"Password generated successfully.\"
          },
          \"data\": {
            \"password\": \"A1b2C3d4E5f6\"
          }
        },
        {
          \"from\": \"assistant\",
          \"value\": \"json{
            \"Thought\": \"I have generated the password 'A1b2C3d4E5f6'
            successfully. Now, I will hash this password using the MD5 algorithm.\",
            \"Action\": \"[MD5 Text Hash(dataString='A1b2C3d4E5f6')]\",
            \"\"}
          }
        ]
      }
    }
```

Figure 19: Nested multiple tools calling task example.