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

Anonymous ACL submission

Abstract

Tool calling enables large language models (LLMs) to interact with external systems, such as APIs and databases, significantly enhancing their capabilities beyond text generation. This functionality is critical for applications like customer support, data analysis, and dynamic content generation. While recent advancements have improved LLM performance in tool invocation tasks, challenges persist, particularly with datasets that rely on simulated or inaccessible APIs and are often limited in geographical diversity. To address these issues, we introduce the International Tool Calling (ITC) dataset, designed specifically for international tool-calling scenarios. The ITC dataset includes 3,571 APIs and 17,540 tool calling tasks, with APIs covering 20 categories and extensive geographical representation from 40 countries. We propose a four-stage pipeline to construct the dataset, incorporating techniques such as bias sampling and tool fusion, and use advanced models to refine queries for high-quality tasks. Experimental results demonstrate significant performance variations between open-source and closed-source LLMs, highlighting the dataset's potential to identify key strengths and weaknesses in tool-calling tasks. Additionally, fine-tuning open-source LLMs using the ITC dataset results in substantial performance improvements, both for in-distribution and out-of-distribution data. Our findings show that the ITC dataset provides a valuable resource for training LLMs in complex international and multiple tools contexts. The data is available at 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 other software tools, thereby extending their capabilities beyond mere text generation. By invoking these tools, LLMs can perform tasks requiring real-time data access, complex computations, or actions outside their training data. This functionality is vital for applications such as automated customer support, data analysis, and dynamic content generation, where the model's ability to access and utilize external resources significantly enhances its performance and utility. Integrating tool calling into LLMs enables more sophisticated, context-aware interactions, making them valuable assets in diverse domains.

In recent years, LLMs have shown remarkable progress in tool invocation tasks, leading to the development of several datasets and frameworks aimed at improving their performance. Key benchmarks include API-BLEND, APIGen, and ToolACE, which focus on API-based function calling, providing diverse APIs for training and evaluation. Other datasets, like Gorilla and ToolLLM, enhance LLM capabilities in real-world API interactions by improving accuracy and reducing issues such as hallucinations. Additionally, datasets such as Seal-Tools, PLUTO, and SciToolBench introduce more complex tool-use scenarios involving multi-step reasoning and domain-specific knowledge. These benchmarks have significantly advanced the development of LLMs capable of understanding and effectively using external tools.

Despite these advancements, several challenges persist. Many datasets rely on simulated APIs, which fail to capture the complexity and variability of real-world tool usage. Although some datasets incorporate real APIs, they often include proprietary or inaccessible endpoints, limiting their practical adoption. Furthermore, many benchmarks are not publicly accessible, restricting their usability for both research and real-world applications. Additionally, most existing datasets are predominantly US-centric, making them less suitable for tasks requiring region-specific information, particularly in a global context. For instance, while Yahoo_Weather, a popular weather API, can provide weather information for Shenzhen, a famous city in China, it cannot retrieve detailed weather data for Nanshan, a district within Shenzhen. This highlights the need for more diverse, publicly available datasets that support international tool-calling scenarios.

To address these challenges, we introduce the International Tool Calling (ITC) dataset, specifically designed for international tool-calling scenarios. We propose a four-stage pipeline for constructing the dataset, which includes collecting APIs from diverse sources, preprocessing the APIs through supplementation and filtering, and applying bias sampling and tool fusion techniques to tackle the long-tail problem of APIs and increase query complexity. Additionally, we refine queries for clarity, relevance, and executability using Claude-3.5-Sonnet and Gemini-1.5-Pro to generate high-quality tool calling tasks. The ITC dataset consists of 3,571 APIs and 17,540 QA pairs, with 15,790 training and 1,750 testing tasks. These APIs span 20 categories, with the largest being Finance, Data, Communication, and Entertainment. The dataset offers extensive geographical coverage, featuring global APIs (64.2%) as well as region-specific APIs, primarily from the US and China, along with long-tail APIs from over 27 countries. It includes 14,295 single tool calling tasks and 3,245 multiple tools calling tasks, with a focus on balancing underrepresented APIs. The ITC dataset serves as a valuable resource for training LLMs in international and multiple tools contexts.

We investigated the performance of 16 open-source LLMs and 8 closed-source LLMs on new testing data, with the experimental results revealing significant performance variations between the two groups across multiple metrics. These findings emphasize the value of the dataset in identifying both strengths and weaknesses in LLM performance, particularly in areas such as handling non-existent tools, missing parameters, and generating incorrect parameters. Additionally, we fine-tuned several open-source LLMs, which had not been previously fine-tuned on other tool-calling benchmarks, using our training dataset. These models showed notable improvements compared to their original versions, both on in-distribution and out-of-distribution (OOD) data. These results demonstrate that our training dataset can effectively enhance model performance for a range of real-world, international tool-calling tasks.

Dataset	# Tools	Source	Access.	# Tasks	Ex.	Lang.
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
ours	**3,573**	**Real**	√	**17,540**	√	**Multi**

Table 1: Summary of existing tool calling datasets. Access.: Accessibility, Ex.: Executable, Lang.: Language, Eng.: English, Sim.: Simulated.

2 Related Work

In this work, we review datasets and frameworks designed to enhance the performance of large language models (LLMs) in tool invocation tasks. These include datasets for API-related tasks, multi-modal tool interactions, and methods for improving tool learning. Table 1 summarizes key datasets and frameworks.

Existing benchmarks cover a range of tool-augmented 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.

Despite these advancements, several limitations persist in existing tool-calling datasets. Many rely on simulated APIs, which may not accurately reflect real-world tool usage, while those utilizing real APIs often include proprietary or inaccessible endpoints, limiting practical adoption. Moreover, more than half of the available benchmarks are not publicly accessible, further restricting their usability. Although ToolBench leverages real APIs and is accessible, some of the APIs sourced from RapidAPI are unavailable. Additionally, most datasets are primarily US-centric, reducing their applicability for retrieving fine-grained local information from other countries.

To address these shortcomings, we introduce a new dataset designed for international tool-calling scenarios. Our dataset aggregates information from a diverse set of publicly available, real international APIs, covering services such as gasoline prices, global and local news, stock markets, and weather conditions across various countries. This broad and diverse coverage enables comparative analysis across regions in fields like economics, media, and environmental sciences. By providing real international APIs along with QA pairs that require region-specific information retrieval, our dataset offers a valuable resource for both academic research and business applications, facilitating deeper insights into global trends and regional differences.

3 Dataset Curation

Our dataset construction follows a four-stage pipeline for minimal human intervention and scalability. First, in **API Collection and Processing**, we use automated tools to gather API documentation. Next, in **Query Generation**, GPT-4o generates detailed instructions for the APIs. In **Query Filtering**, the queries are refined for clarity, relevance, and executability using Claude-3.5-Sonnet and Gemini-1.5-Pro. Finally, in **Question-and-Answer Pair Generation**, GPT-4o generates high-quality Q&A pairs. This pipeline is easily scalable for new APIs, as illustrated in Figure 1.

3.1 API Collection and Processing

API Collection: ToolBench (Qin et al., 2023) is a widely used tool invocation benchmark that selects APIs primarily from RapidAPI. However, most of these APIs originate from the United States, with

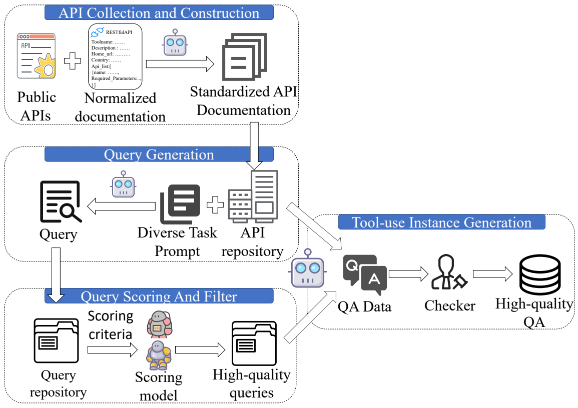

Figure 1: Dataset construction flowchart.

limited representation from other regions, resulting in a lack of global diversity. Additionally, many of these APIs require a paid subscription, restricting accessibility.

To address these limitations, we collected 49,937 real REST APIs from various platforms, spanning domains such as social media, e-commerce, and weather. Our dataset ensures broader geographical representation by sourcing APIs not only from RapidAPI Hub[1] but also from multiple international platforms. These include China's Juhe API[2] and XiaRou API[3], as well as community-curated repositories on GitHub, such as free-api[4] and public-apis[5]. These sources represent just a portion of our efforts to build a globally diverse and accessible API dataset. These APIs are categorized into 20 distinct functional groups, including Entertainment, Finance, Education, and more.

API Supplementation: Since our APIs are sourced from a variety of platforms, ensuring that each one is supported by comprehensive, high-quality documentation—enabling LLMs to accurately interpret the API's functionality, usage, and constraints—can be challenging. Many real-world APIs lack detailed documentation, which exacerbates this issue. To address this, we provide thorough specifications for each API, including clear descriptions of their functionality and well-defined input/output schemas. We also conduct rigorous quality checks, removing incomplete or ambiguous documentation and enriching simpler APIs with additional details. These efforts enhance clarity,

[1] https://rapidapi.com/
[2] https://www.juhe.cn/
[3] https://api.aa1.cn/
[4] https://github.com/fangzesheng/free-api
[5] https://github.com/public-apis/public-apis

improve usability, and ultimately benefit developers by facilitating more efficient and accurate integration.

API Filtering: The initial collection of 49,937 free APIs contained many redundant or low-quality entries. For instance, multiple APIs provided similar functionality, such as weather data, while others suffered from issues like instability, infrequent updates, and poor response accuracy. This reduced the total number of APIs from 49,937 to 5,410. We then conducted a more rigorous screening process, evaluating each API based on key criteria such as stability, update frequency, and response accuracy. As a result, our final dataset includes 3,571 high-quality, free APIs.

3.2 Query Generation

Tool-calling tasks can be categorized into two main types: **Single Tool Calling Task**, which involve calling a single API to accomplish the task, and **Multiple Tools Calling Task**, which require invoking multiple APIs, potentially from different countries. The **Multiple Tools Calling Task** can be further subdivided into three categories: **Repeated multiple tools calling**, where the model makes multiple calls to the same API with different parameters to complete a multi-stage process; **Parallel multiple tools calling**, which involve invoking two or more APIs simultaneously to fulfill the task; and **Nested multiple tools calling**, where the model decomposes the task into steps, invoking APIs in a specific sequence, with the output of one API serving as the input for the next.

Existing benchmarks for generating queries for API calls typically focus on queries in English, with APIs predominantly sourced from the USA. However, in international toll calling scenarios, queries can be in multiple languages, and the APIs may originate from various countries, introducing additional complexity in both language and regional variations. Current benchmarks often overlook the importance of language and location-specific requirements.

In this paper, we design tasks that involve retrieving local information from region-specific APIs. For example, consider a Japanese tourist planning a trip to Lijiang, a popular city in Yunnan Province, China. The tourist would require weather updates and information about local travel destinations from APIs in both Japan and China, presented in Japanese and Chinese, respectively. This scenario underscores the need for cross-lingual and cross-national API interactions, highlighting the challenges and requirements of handling diverse languages and regions in API-based tasks.

Based on the principles outlined above, we designed 36 seed examples covering a range of scenarios. For each case, we first select the desired number of APIs (one or more) and three corresponding examples with the following strategies to improve query quality:

1. **Biased Sampling**. The API collection exhibits a long-tail distribution, with countries such as China and the United States, possessing more advanced internet infrastructures, having a disproportionately higher number of APIs compared to other nations. To address this imbalance, we intentionally generate more instructions for countries with fewer available APIs, thereby reducing sampling bias and ensuring more diverse representation.

2. **Tool Confusion**. To increase the task's complexity, we introduce a challenge for large language models (LLMs) by making it harder for them to differentiate between similar APIs within the same category. This is accomplished by selecting APIs with overlapping functionalities or by including APIs from different countries that might appear similar. We then generate queries that intentionally create ambiguity, potentially misleading the LLM into selecting the wrong API, thus testing its ability to handle nuanced distinctions.

We then prompt GPT-4 to generate the necessary queries, ensuring a diverse and randomized set of outcomes. The output is formatted in JSON, with the "Thought" label capturing the reasoning process in the same language as the query, and the "Action" label indicating the corresponding API calls.

3.3 Query Selection

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. After performing scoring and filtering to select high-quality queries, we reduced the dataset to 17,540 final queries.

Our query selection method consists of two key steps: **Query Scoring** and **Query Selection**. In the

Query Scoring step, we address the limitation of existing datasets, which often lack a fine-grained evaluation standard for assessing the quality of generated queries or question-and-answer pairs. Most existing methods rely on coarse metrics that fail to fully capture the effectiveness and relevance of the instructions. To address this gap, we propose five scoring dimensions: *Relevance*, which evaluates how well the query aligns with the task at hand; *Practicality*, which assesses the feasibility of the instruction in real-world scenarios; *Linguistic Applicability*, which checks for adherence to linguistic norms and cultural context; *Clarity*, which ensures the instruction is clear, concise, and easily understandable; and *Specificity*, which measures the level of detail and focus, reducing ambiguity and enhancing the precision of tool invocation. These scoring dimensions provide a comprehensive evaluation of the instructions from multiple perspectives, ensuring that the final instructions meet high-quality standards across all aspects. To avoid potential model bias, as highlighted in recent work (Zheng et al., 2023), we utilize two independent scoring standards: Anthropic's Claude-3.5-sonnet model and Google's Gemini-1.5-pro model. Both models assign a score ranging from 1 to 5, where 1 indicates a very low quality and 5 indicates the highest quality. Each model evaluates the instructions independently, and filtering and optimization are performed based on the scores provided by both models.

In the **Query Selection** 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 ensures that lower-scoring instructions, which may contain irrelevant or poorly structured content, are effectively excluded from the final dataset. Through this rigorous filtering process, we ensure that the retained instructions adhere to high standards of relevance, practicality, clarity, specificity, and linguistic applicability. This ultimately improves the reliability and usability of the queries for further tasks.

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 developed International Tool Calling (ITC) dataset includes 3,571 APIs and a total of 17,540 question-and-answer pairs, comprising 15,790 training pairs and 1,750 testing pairs.We will introduce the composition of our dataset from two parts: API and query.

4.1 Statistics on APIs

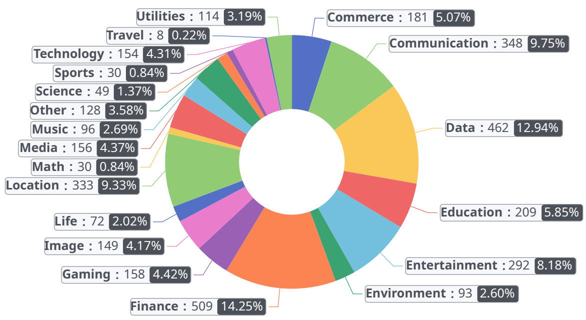

Figure 2: Distribution of tools across 20 categories.

Figure 2 illustrates the distribution of 3,571 APIs across 20 categories. 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 USA. The

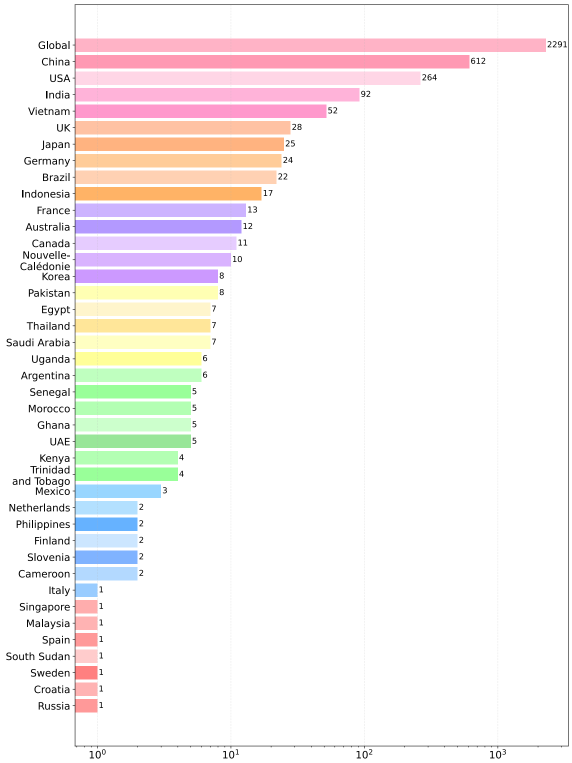

Figure 3: Distribution of tools by countries (Logarithmic Scale).

second category is **country-specific** APIs, which offer information tailored to a particular country and its language, such as local weather updates and news. As shown in Figure 3, **global** APIs account for 2,291 samples (64.2%). China and the United States contribute the majority numbers of **country-specific** APIs. Finally, **Long-tail** APIs, originating from over 27 other countries and regions, contribute 11.31% of the dataset.

4.2 Statistics on Tasks

Our dataset consists of 17,540 tasks, including 14,295 single-tool calling tasks and 3,245 multiple tools calling tasks. 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 taks, 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 included both open-source and closed-source LLMs in our experiments. The open-source models, which are freely accessible for research and development, include Qwen2.5(Yang et al., 2024), Hammer2(Lin et al., 2024b), Functionary-small-v3.1, ToolACE-8B(Liu et al., 2024b), Watt-tool-8B, Yi-1.5-9B-Chat-16K(Young et al., 2024), glm-4-9b-chat(GLM et al., 2024), and Phi-4(Abdin et al., 2024). These models exhibit diverse performance across various task categories, such as tool invocation and argument extraction.In contrast, the closed-source models consist of GPT4o, GLM-Zero-Preview, Gemini-2.0(Team et al., 2023), Claude-3.5-Sonnet, Deepseek-V3(Liu et al., 2024a), and Deepseek-R1(Guo et al., 2025).

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

We evaluate the experimental results using four metrics: **Tool Selection (P/R/F1)** measures the model's ability to identify appropriate tools from candidates, focusing on tool localization accuracy, computed using precision, recall, and F1-score. **Tool Invocation (P/R/F1)** evaluates the model's understanding of tool parameters and the completeness of structured information extraction, also using precision, recall, and F1-score through triple matching. **Language Matching Accuracy (LM)** quantifies how closely the output adheres to the target language requirements, specifically assessing whether the "thought" label is in the same language as the query, computed with the `langid` library. Finally, **Format Matching Accuracy (FM)** measures the model's ability to conform to the expected JSON format needed for successful tool invocation.

5.3 Evaluation on testing datasets

Main results: The experimental results presented in Table 2 highlight the performance differences between open-source and closed-source models on tool invocation tasks. In summary, closed-source LLMs generally outperform open-source LLMs. In

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	**88.90**	**88.03**	**88.30**	**76.33**	**73.46**	**74.31**
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	**96.73**	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	**100**	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	**99.83**	76.47	75.21	75.55	71.69	70.38	70.71
GPT4o	**97.95**	99.83	**88.95**	**89.48**	**89.01**	**82.18**	**81.57**	**81.57**
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: Experimental results on our testing dataset (%).

terms of **Language Matching Accuracy (LM)**, scores range from 74% to over 97%, with open-source models like Qwen2.5-7B-Instruct (90.51%) and closed-source models like GPT4o (97.95%) exhibiting strong linguistic consistency. Regarding **Format Matching Accuracy (FM)**, most models show solid adherence to expected output formats, ranging from 4.56% to 99.89%. Open-source models such as Deepseek-R1 (100%) and Qwen2.5-Coder-32B (99.14%) excel in maintaining the required JSON format, while models like watt-tool-8B and ToolACE-8B show lower performance in this aspect.

Performance also varies in **Tool Selection** and **Tool Invocation** metrics. Models like GPT4o lead in **Tool Selection (P/R/F1)**, demonstrating high precision, recall, and F1 scores, while open-source models like watt-tool-8B also perform well. However, models like Functionary-v3.1 show limited capabilities in selecting appropriate tools. Similarly, in **Tool Invocation (P/R/F1)**, GPT4o and watt-tool-8B excel, indicating strong tool invocation capabilities, while models such as Hammer2.1-7b struggle to generate and structure effective tool invocations.

The difference between Tool Selection and Tool Invocation reflects a model's ability to choose the right tool and generate the correct invocation parameters. Models like GPT4o show minimal discrepancy, excelling in both aspects. In contrast, models with a larger gap, such as Hammer2.1-7b, struggle with generating proper parameters for selected tools, highlighting areas for improvement in parameter generation and structural output. This discrepancy is a key performance indicator for tool-based tasks.

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	16.42	41.45
Qwen2.5-Coder-3B	38.48	51.51	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	9.34
Hammer2.1-7b	0.70	91.56	7.74	17.18	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	2.11	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	0.43	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	2.82	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	11.29
GPT4o	47.16	49.72	3.12	53.67	21.22	25.10
GLM-Zero	37.63	58.09	4.28	48.89	13.27	37.83
gemini-2.0-flash	30.93	62.13	6.95	50.42	29.50	20.08
gemini-2.0-pro	0	85.64	14.36	47.31	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. in tool invocation: generating extra parameters.

Error analysis: Table 3 presents an analysis of all LLMs in tool selection and invocation. Across all models, tool selection errors primarily consist of hallucinating non-existing tools and missing required tools. Notably, models like Qwen2.5-72B-Instruct and Hammer2.1-7b show the highest and lowest hallucination rates, respectively, with significant variation in the missing tools rate, as seen in glm-4-9b-chat and Deepseek-V3. Extra tool calls (Ex. in tool selection) also vary, with models like Qwen2.5-Coder-32B generating higher rates compared to Deepseek-V3. For tool invocation errors, models display notable discrepancies in generating incorrect parameters, with Hammer2.1-7b having a high rate of 64.26%. Additionally, models like GPT4o-mini and gemini-2.0-flash exhibit a more balanced performance in terms of missing parameters and extra parameters. Overall, the main challenges highlighted are tool hallucination, missing parameters, and generating incorrect invocation parameters, reflecting areas for potential improvement across different models.

Model Name	LM	FC	Tool Selection			Tool Invocation		
			P	R	F1	P	R	F1
Qwen2.5-7B-Instruct	96.89(**+6.38**)	99.77(**+3.12**)	97.72(**+43.64**)	98.08(**+45.02**)	97.76(**+44.58**)	90.64(**+47.88**)	90.55(**+47.18**)	90.34(**+47.63**)
Qwen2.5-Coder-7B	97.41(**+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(**+18.01**)	97.92(**+22.08**)	97.48(**+49.96**)	89.78(**+28.88**)	89.50(**+27.73**)	89.36(**+48.03**)
Qwen2.5-Coder-3B	97.29(**+13.03**)	99.79(**+10.54**)	97.64(**+48.72**)	97.89(**+48.88**)	97.64(**+48.88**)	90.25(**+51.76**)	90.26(**+51.43**)	89.96(**+51.53**)

Table 4: Results on our testing dataset by fine-tuned LLMs (%), with values in brackets showing the improvement from the original models.

Model	**Nexus Raven**		**Seal-Tools**		**Tool-Alpaca**	
	Tool Selection	**Tool Invocation**	**Tool Selection**	**Tool Invocation**	**Tool Selection**	**Tool Invocation**
Qwen2.5-7B-Instruct	90.57(**+25.75**)	59.97(**+10.23**)	89.91(**+24.00**)	76.16(**+17.76**)	77.05(**+18.10**)	49.96(**+9.85**)
Qwen2.5-Coder-7B-Instruct	90.99(**+20.44**)	68.04(**+17.76**)	89.57(**+22.22**)	78.04(**+18.92**)	77.34(**+14.91**)	50.87(**+8.23**)
Qwen2.5-3B-Instruct	81.03(**+6.50**)	57.14(**+2.78**)	90.32(**+23.26**)	76.76(**+20.02**)	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: Results on three benchmark testing dataset by fine-tuned LLMs (%), with values in brackets showing the improvement from the original models.

5.4 Evaluation for Fine-tuning

In this experiment, we fine-tuned four versions of Qwen 2.5—Qwen2.5-7B-Instruct, Qwen2.5-Coder-7B, Qwen2.5-3B-Instruct, and Qwen2.5-Coder-3B—on our training dataset. These models had not previously been fine-tuned on any tool-calling datasets, allowing us to evaluate the impact of our training dataset on enhancing the tool-calling capabilities of open-source LLMs.

Testing results on our testing dataset: The results, shown in Table 4, reveal substantial improvements in both tool selection and tool invocation after fine-tuning the large language models (LLMs). Tool invocation gains reached up to 51.76%, with the smaller 3B models performing almost as well as their larger 7B counterparts, highlighting the effectiveness of our training dataset. The evaluated models include Qwen2.5-7B-Instruct, Qwen2.5-Coder-7B, Qwen2.5-3B-Instruct, and Qwen2.5-Coder-3B. Across all models, significant improvements were observed, especially in tool selection and invocation tasks. For example, Qwen2.5-7B-Instruct showed a 47.63% increase in tool selection F1 score and a 47.88% improvement in tool invocation F1 score. Notably, Qwen2.5-Coder-3B achieved the highest improvement in tool invocation, with a 51.53% increase in F1 score. Remarkably, Qwen2.5-Coder-3B achieved performance levels comparable to Qwen2.5-7B-Instruct, further demonstrating the success of our training dataset in enhancing the models' tool-calling capabilities.

Testing results on OOD data: To evaluate the robustness and generalization of our fine-tuned models, we tested them on three OOD datasets: Nexus Raven(team, 2023), Seal-Tools(Wu et al., 2024), and Tool-Alpaca(Tang et al., 2023a). The results, shown in Table 5, reveal significant improvements across all four models. Four fine-tuned versions of Qwen2.5 exhibited notable enhancements in tool selection and invocation, in terms in F1, with improvements reaching up to 25.75% for tool selection and 18.10% for tool invocation. These findings highlight the effectiveness of our training dataset in boosting model performance on various tool-calling tasks.

6 Conclusion

In this paper, we addresses the pressing need for a more diverse, globally-oriented dataset to support the development of LLMs' tool-calling capabilities. By introducing the International Tool Calling (ITC) dataset, we provide a comprehensive resource for training and evaluating LLMs on international and multiple tools calling scenarios. Our dataset, which spans a wide range of API categories and includes both global and region-specific APIs, effectively tackles the challenges of long-tail API representation and the complexities of multiple tools calling interactions. The experimental results underscore the utility of the ITC dataset in identifying key performance issues in LLM tool invocation, such as handling missing or incorrect parameters, and demonstrate the potential for significant performance improvement through fine-tuning. These findings highlight the promise of our dataset in advancing LLMs' ability to interact with international APIs, and suggest promising directions for future research.

Limitations

While our work offers significant advancements, several limitations warrant further investigation. First, although our dataset prioritizes geographical diversity, certain regions (e.g., Africa and parts of Asia) remain underrepresented, potentially limiting the model's ability to handle nuanced cultural or regulatory contexts in these areas. Second, the dataset focuses exclusively on REST APIs, leaving other tool types (e.g., SOAP APIs or database connectors) unexplored, which may restrict applicability in heterogeneous tool ecosystems. Third, while our automated pipeline ensures scalability, it may inadvertently propagate biases or errors from synthetic instruction generation, particularly in low-resource languages. Additionally, error analysis revealed persistent challenges in handling nested tool calls and parameter hallucination, suggesting the need for stronger semantic validation frameworks. Finally, the reliance on free APIs introduces potential instability due to service deprecation or rate limits, which could affect long-term reproducibility. Finally, the dataset with advanced difficulty is required to boost the tool calling capabilities for open source LLMs. Addressing these limitations will be critical for future work to achieve truly robust and universal tool-calling systems.

Acknowledgments

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

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, Yuanzhi 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. Sciagent: Tool-augmented language models for scientific reasoning. *arXiv preprint arXiv:2402.11451*.

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

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *Preprint*, arXiv:2305.15334.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Preprint*, arXiv:2302.04761.

Simranjit Singh, Michael Fore, and Dimitrios Stamoulis. 2024. Evaluating tool-augmented agents in remote sensing platforms. *arXiv preprint arXiv:2405.00709*.

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

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

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Nexusflow.ai team. 2023. Nexusraven: Surpassing the state-of-the-art in open-source function calling llms.

Chenyu Wang, Weixin Luo, Qianyu Chen, Haonan Mai, Jindi Guo, Sixun Dong, Zhengxin Li, Lin Ma, Shenghua Gao, et al. 2024. Tool-lmm: A large multi-modal model for tool agent learning. *arXiv preprint arXiv:2401.10727*.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: self-instruct tool learning dataset for agent tuning and detailed benchmark. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 372–384. Springer.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. 2024a. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024b. Toolsword: Unveiling safety issues of large language models in tool learning across three stages. *arXiv preprint arXiv:2402.10753*.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024c. Rotbench: a multi-level benchmark for evaluating the robustness of large language models in tool learning. *arXiv preprint arXiv:2401.08326*.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.

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

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 4: Query generation prompt for single tool calling tasks.

A 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 4.

B 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 5, 6, and 7.

C Query Scoring

Figure 8 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 9 shows the prompt for LLMs to evaluate the query.

```
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 5: Multiple tool repeated calls.

```
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 6: Multiple tool parallel calls.

D Data Examples

Figure 10 illustrates an example of the Google Translate API. Figure 11 provides an example of a single tool calling task, while Figure 12 demonstrates a repeated multiple tools calling task. Figure 13 shows an example of a parallel multiple tools calling task, and Figure 14 presents an example of a nested multiple tools calling task.

```
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 7: Multiple tool nested calls.

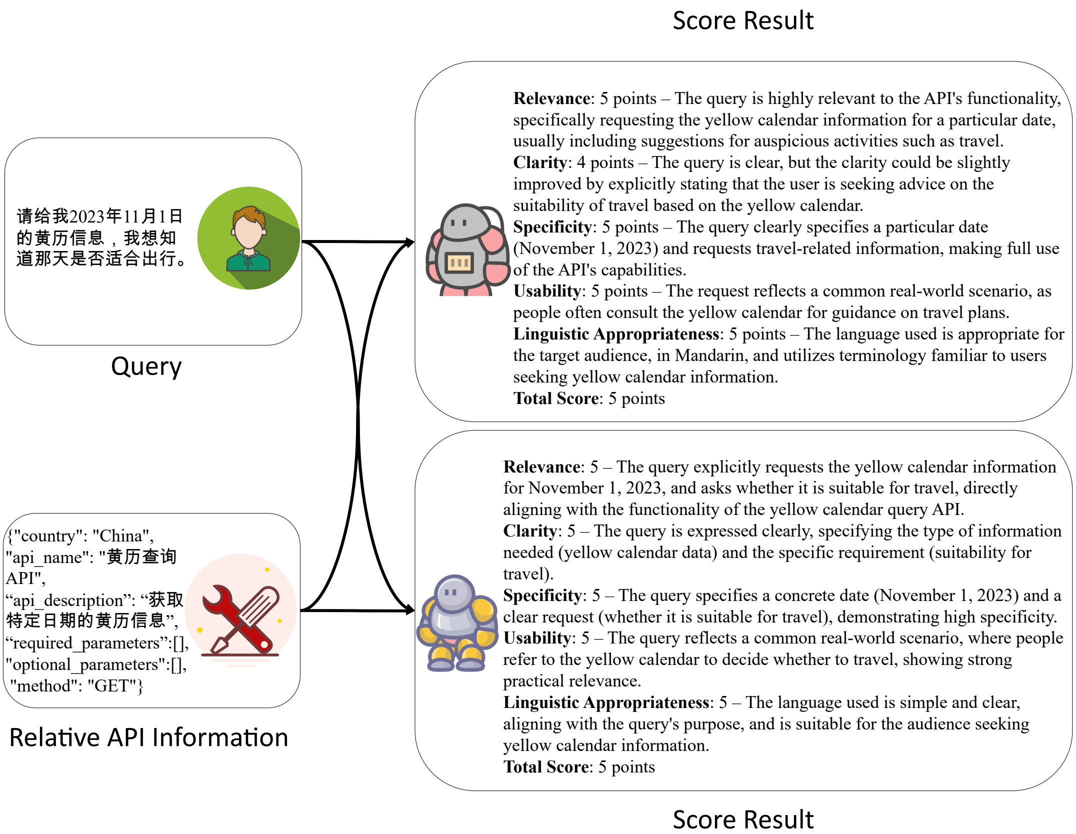

Figure 8: The query scoring process.

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 9: 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 10: 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 11: 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 12: 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 13: 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 14: Nested multiple tools calling task example.