PracTool: Advancing Tool Usage in LLMs through User Intent Recognition and Decision-Making

Anonymous ACL submission

Abstract

Recently, tool-enhanced large language models (LLMs) have gained attention for their exceptional ability to acquire up-to-date information and solve specialized problems. Most research focuses on improving LLMs' accuracy in invoking tools for queries with explicit intentions, involving tool selection, parameter filling, and formatted outputs. However, strong tool-calling capabilities alone are insufficient for practical applications, as (1) many real-life queries don't require tool invocation, thus LLMs need robust decision-making skills; and (2) when implicit invocation intentions exist, LLMs must recognize and select appropriate tools. To address these challenges, we introduce **PracTool**, a user intent-driven tool-use framework that encompasses data construction, model training, and evaluation. We categorize four scenarios based on the clarity of tool intentions and their invocation requirements, employing various data generation processes for comprehensive testing in decision-making and implicit intention reasoning. Additionally, we propose the CoTx reasoning method, enhancing the model's capabilities based on the Chain of Thought (CoT) approach. Using PracTool, we fine-tune Qwen to develop ToolQwen, equipping it with CoTx. Extensive experiments show that ToolQwen's decision-making and tool-calling capabilities are comparable to or exceed GPT-4 across diverse scenarios, bringing the model closer to practical deployment.

1 Introduction

Recent advancements in large language models (LLMs) have been both significant and rapid (Zhao et al., 2023; Kaddour et al., 2023). While LLMs exhibit impressive capabilities across a range of applications, they continue to face certain limitations, such as scientific calculations (Ahn et al., 2024) and real-time information retrieval (Fan et al., 2024; Gao et al., 2023). Tool learning presents a promising solution by enabling LLMs to effectively leverage external tools (Zhiruo Wang, 2024; Qu et al., 2024). Through the strategic use of these real-world tools, LLMs can: (1) effectively bridge existing capability gaps; for instance, when faced with complex calculations, they can utilize a scientific calculator tool to generate accurate results (Kadlčík et al., 2023); (2) access the most current information, such as obtaining the latest exchange rates among various currencies via an exchange rate querying tool (Xu et al., 2023); and (3) accomplish practical tasks, such as flight bookings, thereby evolving into more capable and intelligent agents (Xi et al.; Sumers et al., 2023).

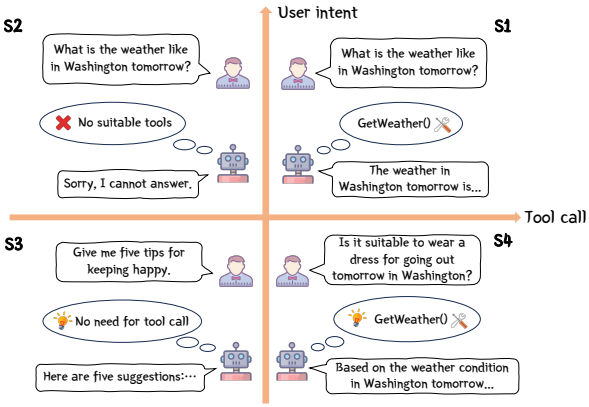

Figure 1: Illustration of intent-driven tool-use framework. We consider four scenarios based on the clarity of user intent and the necessity of tool calls, represented by S1-S4 for each respective scenario.

Despite considerable efforts in existing research on tool usage, most studies are confined to scenarios where the user's query explicitly indicates a tool invocation intent (Qin et al., 2023; Tang et al., 2023; Schick et al., 2024). This limitation creates a substantial gap when considering the diverse range of real-world situations. As illustrated in Figure 1, we can categorize tool usage and user query intent into four distinct scenarios along two axes. Above the horizontal axis, where users exhibit a clear in-

tent for tool invocation: (1) The model should accurately invoke the appropriate tool, which is the primary focus of most research. (2) The model refrains from making a tool invocation, potentially signaling either the absence of suitable tools or a failure to provide an accurate response. Below the horizontal axis, where users lack a clear tool invocation intent: (3) The model successfully identifies situations where no tool invocation is necessary. (4) The user's query contains an implicit intent for tool invocation, requiring the model to recognize this intent and subsequently provide the appropriate tool invocation.

In contrast to existing researches (Qin et al., 2023; Paranjape et al., 2023; Li et al., 2023; Wang et al., 2024) that are limited to generating diverse tool-use data with clear intentions, we propose an intent-driven tool use framework named PracTool. This framework aims to enhance tool invocation capabilities across various user intent contexts and encompasses data construction, model training, and comprehensive evaluation. As illustrated in Figure 2, we first develop diverse data generation pipelines to create a rich dataset using ChatGPT(*gpt-3.5-16k-turbo*) across various scenarios. Subsequently, we fine-tune the Qwen2-7B (Yang et al., 2024) model to create ToolQwen, thereby enhancing the tool-use capabilities of the smaller model through knowledge injection. Recognizing the critical importance of accurately identifying users' tool usage intents, we further propose a refined Chain of Thought (CoTx) framework based on the foundational CoT approach (Wei et al., 2022; Feng et al., 2024). CoTx functions similarly to a binary decision tree (Yao et al., 2024), guiding the model at each branching point to evaluate the necessity of tool usage, identify user intent, and select the appropriate tools. A model empowered by CoTx is better equipped to make correct tool calls across a variety of scenarios, thereby significantly enhancing its reasoning capabilities. In our evaluation framework, we evaluate both the model's positive capabilities and its errors in a unified manner. Utilizing fine-grained and comprehensive metrics allows us to accurately reflect the model's tool-use performance. Experimental results demonstrate that smaller models trained with Supervised Fine-Tuning (SFT) and enhanced with CoTx not only achieve superior performance but also significantly outpace the current state-of-the-art model, GPT-4.

In summary, our main contributions include:

- We define tool invocation scenarios that are more aligned with real-world applications by emphasizing the significance of user intent and the necessity of tool invocation. Through diverse data construction pipelines, we have created challenging and high quality tool-use datasets across different scenarios.

- We designed a finely controlled Chain of Thought process that significantly enhances the model's reasoning capabilities even in a training-free context, enabling better awareness of tool invocation under a variety of user queries.

- Extensive experiments demonstrate that, compared to prompts, SFT can enhance the model's performance in diverse scenarios. Even the 7B parameter Qwen2 model can substantially outperform the most advanced closed-source models.

2 PracTool

2.1 Tool Collection

Considering that tool usage needs to cover the vast majority of common requirements, we manually define and develop 15 of the most frequently used tools, thereby forming the tool pool $\mathbb{S}_{api} = \{API_1, API_2, ...\}$. These include tools for real-time information retrieval (such as exchange rates and weather queries), domain-specific capabilities (such as scientific calculators and translation), as well as task completion tools (like playing music). Details of the tool definitions can be found in Appendix E.

2.2 Dataset Construction

2.2.1 Clear Intent Queries(S1&S2)

When users exhibit a clear intent for tool invocation, we generate data for two scenarios through an automated pipeline.

First, for a specific tool API_i, we prompt ChatGPT to understand the API's definition and functionality, subsequently generating potential query-call pairs that could invoke the tool. Simultaneously, we use a discriminative prompt to have GPT-4 check the generated query-call pairs for their validity and filter out unreasonable examples. Next, for each query, we randomly sample a candidate toolset $\mathbb{S}_{sub}$ from the tool pool $\mathbb{S}_{api}$. At this point, there are two possibilities:

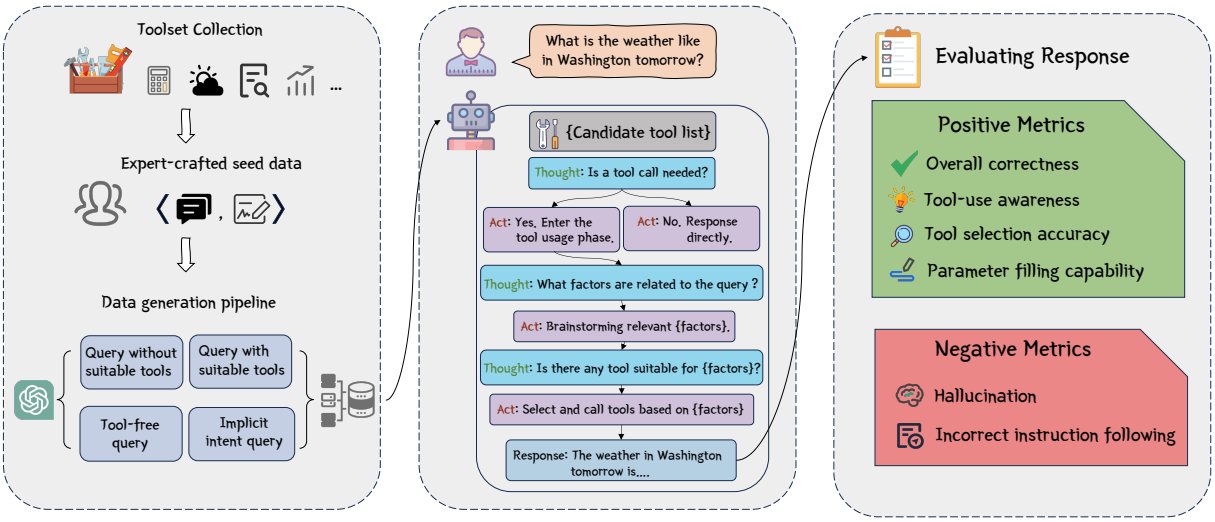

Figure 2: The overall architecture of PracTool encompasses data generation, model training and reasoning, as well as a comprehensive evaluation framework.

- If API_i is included in $\mathbb{S}_{sub}$, the pair $< \mathbb{S}_{sub}, query >$ represents a reasonable tool invocation. Thus, we classify $< \mathbb{S}_{sub}, query + call >$ as a sample arising from the "Query with Suitable Tools" scenarios.

- If API_i is not included in $\mathbb{S}_{sub}$, the query cannot be reasonably addressed using the tools in the candidate toolset, and we categorize $< \mathbb{S}_{sub}, query + Nocall >$ as a sample arising from the "Query without Suitable Tools" scenarios.

In this way, we can construct datasets for both quadrants within a single pipeline.

2.2.2 Tool-free Queries(S3)

When a user's query neither indicates an intent for tool invocation nor requires a tool to provide a response, we construct "Tool-free" queries with varying levels of difficulty based on two primary aspects:

(1) Simple Samples: We filter queries from online users that do not involve tool invocation and label these as "No Call." These queries typically exhibit a significant semantic distinction from those requiring tool invocation, making it relatively straightforward for the model to identify and classify these samples.

(2) Difficult Samples: For a tool API_i, we begin by manually curating a set of high-frequency keywords $W = \{w_i\}_{i=1}^{N}$ associated with tool invocation. For instance, when examining a gold price inquiry tool, the high-frequency keywords might include terms like $\{gold, gold\ price, ...\}$. We then prompt GPT-4 to generate queries that incorporate these high-frequency keywords while intentionally avoiding the invocation of api_i. An example of such a query would be, "What factors influence gold prices?" Despite containing high-frequency keywords, these queries do not trigger API_i invocation, thereby presenting a greater challenge to the model's semantic comprehension and reasoning capabilities. We refer to these as difficult samples.

2.2.3 Implicit Intent Queries(S4)

As shown in Figure 1, certain user queries may not explicitly convey an intent for tool invocation, yet they still necessitate such invocation to yield more effective responses. Given the challenges associated with using natural language or predefined rules to assist large models in comprehending implicit intent scenarios and generating appropriate queries, we propose a pipeline focused on "follow-up questions" to facilitate the generation of implicit intent data.

As illustrated in Figure 3, the process for handling a tool API_i involves several steps: (1) Data Sampling: Sample data pairs <*query,call*> from the clear intent data constructed in Section 2.2.1. (2) Keyword Extraction: We extract high-frequency keywords W associated with API_i, as detailed in Section 2.2.2. (3) Follow-Up Question Generation: We prompt the large language model (LLM) to generate a "follow-up question" based on the original query. This follow-up question is semantically aligned with the original query and occurs within a continuous dialogue context, subtly inheriting the explicit intent of the original query. (4) Parame-

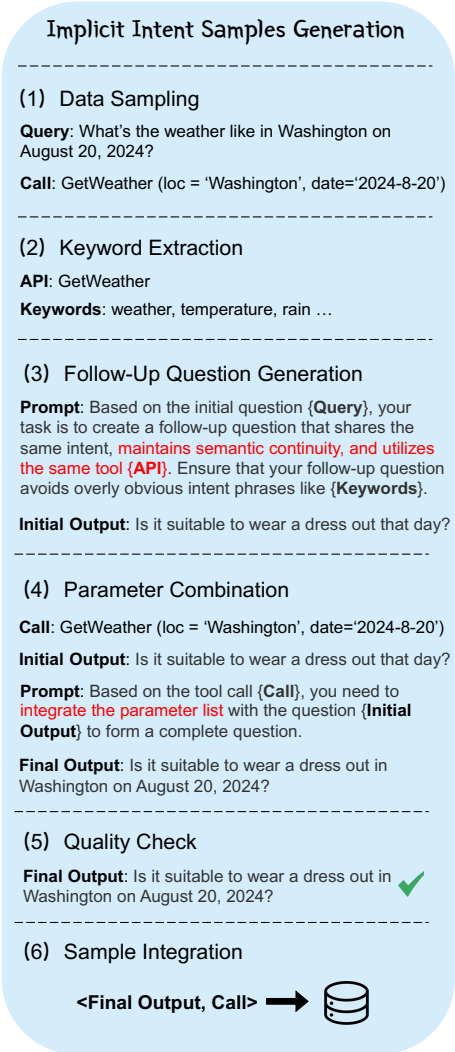

Figure 3: The data construction process for queries with implicit intent.

ter Combination: We integrate the parameters of the call with the follow-up question to address any missing information. (5) Quality Check: We conduct manual data quality checks on the follow-up questions, and filter out the implicit intent queries that meet the requirements. (6) Sample Integration: For the follow-up questions that meet quality criteria, we merge *<Final Output, Call>* to create a cohesive piece of implicit intent questioning sample. In Section 2.4, we address the significant challenges posed by implicit intent questioning in terms of the existing LLM's awareness of tool invocation. We also discuss strategies for enhancing the LLM's capabilities to invoke tools effectively in scenarios involving implicit intent.

2.3 Data Summary

Ultimately, we have built a diverse dataset that together forms PracTool. PracTool covers a total of 4 practical scenarios, including 15 tools and 9,291 samples. Detailed statistics of the samples from various fields can be found in Appendix A.

2.4 CoTx

Considering that diverse scenarios require the model to possess stronger decision-making abilities compared to directly providing tool invocation results, we propose an additional reasoning method based on Chain of Thought (CoT): CoTx, which aims to enhance the model's performance during the reasoning stage. CoTx involves the model following these steps:

- **Does the current query require tool invocation?** If the user's question is within the model's own knowledge base or not related to tool calls (S3), then the model decides not to call any tools and responds directly. Otherwise, it will proceed with the following thinking.
- **What factors are needed to answer the current query?** In this step, we hope the model will brainstorm to associate as many factors as possible related to the current query, forming a set of factors $\mathbb{F} = \{f_1, f_2 ... f_n\}$. This should adequately cover the tool call requirements for both obvious intentions (S1, S2) and implicit intention scenarios (S4).
- **What tools are suitable for the current factors?** Finally, the model relies on the set of factors $\mathbb{F}$ derived from the previous step to select appropriate tools from the candidate tool list and provide the final response based on the tool selection results.

By combining this thought process, the model ultimately provides a response. As shown in Figure 2, CoTx enables the model to think through diverse scenarios by deciding on tool invocation, associating relevant factors, and identifying suitable tools, thus fostering smarter decision-making capabilities.

3 Experiments

3.1 Setup

We utilize datasets derived from the 15 high-frequency tools outlined in Section 2, complemented by data constructed across the four quadrants. To enhance clarity, we have systematically organized the data from each quadrant in Table 4,

which provides detailed sample counts for each dataset.

3.2 Evaluation Metric

Considering the diverse scenarios covered by our dataset, we have constructed a comprehensive set of evaluation metrics from both positive and negative perspectives. In the main experiments, we additionally focus on two key metrics: tool invocation ability and tool invocation awareness. Let the total number of samples be denoted as N. Tool invocation ability is quantified using the formula: $P_{Corr} = \frac{Corr}{N}$, where $Corr$ represents the number of samples in which the model accurately outputs the tool name and completely fills in the parameters correctly. Conversely, we assess the model's tool invocation awareness using: $P_{Awar} = \frac{Awar}{N}$, where $Awar$ indicates the number of samples in which the model correctly determines whether to invoke the tool or not.

3.3 Baselines

Currently, state-of-the-art LLMs, such as ChatGPT and GPT-4, serve as our strongest baselines. To compare against the original models, we also consider LLaMA2-7B/13B-chat and Qwen2-7B/13B-chat models. For the baseline models, we test their original capabilities using a zero-shot approach, while also utilizing In-Context Learning (ICL) with 3 randomly sampled samples, and Chain of Thought (CoT) methods to enhance their reasoning abilities and performance. Additionally, we partitioned the dataset into training and testing sets. We fine-tuned the Qwen2-7B-chat model using LoRA (Hu et al., 2021) on the training set, resulting in the development of ToolQwen. Detailed training configurations can be found in Appendix B.

4 Main Results

The main experimental results are summarized in Table 1, and we highlight the following key findings.

No Base Model Achieves Good Performance Across Diverse Scenarios. No baseline model demonstrated strong performance across varied scenarios. The most advanced model, GPT-4, achieved only 66.93% accuracy in standard tool invocation contexts. In implicit intent situations, such as S4, GPT-4 exhibited significantly reduced performance. Similar findings were observed with models like LLaMA and Qwen. On one hand, smaller models tend to exhibit weaker adherence to instructions, which impedes their ability to conform to the required API call formats. On the other hand, the diversity of scenarios necessitates enhanced semantic understanding and decision-making capabilities from the models, presenting substantial challenges even for GPT-4.

Smaller Models Struggle with Refusal to Invoke Tools. Smaller models demonstrated inferior performance in scenarios requiring the refusal to invoke tools (S2 and S3) compared to GPT-3.5 and GPT-4. A likely contributing factor is that larger models, which have been trained on more extensive datasets, possess significantly better understanding and matching capabilities for tools and queries. Furthermore, in the challenging refusal samples of S4, which contained high-frequency tool usage keywords, larger models were also able to recognize these scenarios more effectively.

ICL and SFT Provide Significant Gains. Both ICL and SFT have demonstrated considerable advantages for the models. Under both methodologies, we observed significant performance improvements. For example, ICL with three randomly selected samples brought an average improvement of 15.47% in P_{Corr} and a 14.17% increase in P_{Awar} for GPT-3.5-turbo. Similarly, several studies (Qin et al., 2023) have shown that employing semantic similarity retrieval to provide more precise ICL samples can yield additional enhancements. We believe that applying this approach will also benefit our experiments. Notably, SFT particularly enhanced model performance, allowing even the 7B model to significantly outperform all other models. SFT facilitated smaller models in achieving well-balanced results across various scenarios, indicating that when practical deployment is required, SFT can offer substantial benefits.

CoTx Can Significantly Enhance the Decision-making Capabilities. The results clearly indicate that when models are equipped with a refined reasoning process, all exhibit significant improvements. Moreover, as a reasoning methodology, CoTx can be directly adapted to closed-source models or those that have undergone SFT, leading to further enhancements in performance. This approach offers a convenient means for models to operate effectively across diverse tool invocation scenarios, thereby facilitating practical deployment in real-world applications.

Setting	Models	S1		S2		S3		S4	
		P_{Corr}	P_{Awar}	P_{Corr}	P_{Awar}	P_{Corr}	P_{Awar}	P_{Corr}	P_{Awar}
Zero-shot	LLaMA2-Chat-7B	24.42	66.74	38.80	38.80	41.43	41.43	5.83	10.17
	LLaMA2-Chat-13B	32.35	70.98	42.84	42.84	43.76	43.76	14.51	26.66
	Qwen2-Chat-7B	29.94	67.78	37.72	37.72	41.45	41.45	12.93	19.87
	Qwen2-Chat-13B	39.55	71.13	41.15	41.15	44.37	44.37	20.04	31.13
	GPT-3.5-turbo	55.82	91.14	52.57	52.57	41.95	41.95	31.48	67.79
	GPT-4	66.93	98.12	81.14	81.14	96.60	96.60	58.84	85.96
ICL	LLaMA2-Chat-7B	32.31	71.58	49.88	49.88	45.54	45.54	9.78	20.01
	LLaMA2-Chat-13B	45.52	88.89	47.74	47.74	46.68	46.68	21.19	40.53
	Qwen2-Chat-7B	35.57	75.55	43.35	43.35	41.14	41.14	15.45	19.98
	Qwen2-Chat-13B	42.20	78.87	36.66	36.66	55.41	55.41	27.89	48.85
	GPT-3.5-turbo	57.40	91.89	65.54	65.54	42.25	42.25	42.09	88.48
	GPT-4	69.96	98.85	84.47	84.47	**97.76**	**97.76**	63.31	92.11
CoTx	LLaMA2-Chat-7B	30.87	72.57	51.59	51.59	44.83	44.83	11.08	27.79
	LLaMA2-Chat-13B	45.05	89.20	49.16	49.16	48.63	48.63	25.71	41.87
	Qwen2-Chat-7B	38.28	75.62	41.45	41.45	42.08	42.08	17.93	21.14
	Qwen2-Chat-13B	43.45	79.32	39.14	39.14	55.15	55.15	27.25	50.42
	GPT-3.5-turbo	57.72	93.61	67.09	67.09	44.16	44.16	40.26	90.36
	GPT-4	68.50	97.74	86.04	86.04	97.15	97.15	64.82	92.92
Fine-tuning	ToolQwen	**82.00**	**99.69**	**89.94**	**89.94**	77.49	77.49	**87.19**	**99.64**

Table 1: Overall tool-use performance. P_{Corr} measures the rate of complete matching between the model's predictions and the answers, while P_{Awar} assesses whether the model is aware of the correct tool usage. Since S2 and S3 do not involve tool calls, P_{Corr} and P_{Awar} are identical in both scenarios. Best overall results are **bold-faced**, and best results under each setting are underscored.

5 Analysis

In this section, we assess the model's strengths and the errors it has encountered through a more detailed analysis of the metrics.

5.1 Tool selection and parameter filling capability

We begin by evaluating the model's abilities in tool selection and parameter filling. Specifically, we selected the most advanced models, GPT-4 and ToolQwen, to assess different reasoning methods in scenarios requiring tool invocation, specifically in S1 and S4. Note that tool selection and parameter filling abilities are only measured in responses that correctly parse out the tool invocation, and we use N_{prop} to denote the number of such samples. We measure the model's tool selection ability using $P_{Name} = \frac{Name}{N_{\text{prop}}}$, where $Name$ denotes the number of samples with the correct tool name. Additionally, recognizing that the answers for parameters are often not unique (e.g., "New York" and "New York City" can be seen as equivalent), we evaluate the model's parameter filling ability (PFA) based on Normalized Levenshtein Distance(LD) (Huang et al., 2024). For a parameter arg, let its gold answer be s and the model's prediction be t. The PFA is measured using the following formula:

$$Normalized\ LD(s,t) = \frac{EditDis(s,t)}{max(len(s), len(t)}$$

$$\text{PFA} = \frac{1}{N_{\text{prop}}} \sum_{i}^{N_{\text{prop}}} 1 - NormalizedLD(s_i, t_i) \quad (1)$$

Intuitively, higher values of P_{Name} and PFA indicate stronger capabilities of the model in tool selection and parameter filling. The experimental results for these two metrics are presented in Table 2. Based on the experimental findings, we outline the following key observations:

ICL Significantly Enhances Tool Selection and Parameter Optimization Capabilities. The introduction of in-context learning markedly enhances the model's abilities in tool selection and

Model	Methods	S1		S4	
		P_{Name}	PFA	P_{Name}	PFA
GPT-4	zero-shot	97.04	94.45	78.70	92.67
	ICL(3-shot)	97.83	98.84	86.33	95.52
	CoTx	97.53	95.21	91.84	91.34
	CoTx+ICL	98.78	98.45	92.10	95.56
ToolQwen	zero-shot	99.23	98.10	90.45	96.62
	ICL(3-shot)	99.29	99.31	91.87	96.69
	CoTx	99.25	98.50	92.03	97.16
	CoTx+ICL	99.68	99.44	95.80	98.13

Table 2: Evaluation of Tool Selection and Parameter Filling.

parameter filling. In comparison to zero-shot, the inclusion of in-context samples resulted in a significant improvement, particularly in parameter filling. In both scenarios, GPT-4 exhibited an average increase of 4.2%. ICL effectively elucidates the semantics of the API through contextual samples and usage examples, thereby facilitating improved model performance.

CoTx Substantially Enhances the Model's Tool Selection Ability in Scenarios with Implicit Intentions CoTx effectively leverages associations and reasoning related to various factors in the queries, which leads to a substantial enhancement in the model's ability to discern intentions when invoking tools under implicit expressions. GPT-4 achieves a gain of 16.7%(78.7 $\rightarrow$ 91.84) in this context. Furthermore, this approach aids in the accurate selection of the appropriate tool.

Integration of CoTx and ICL Yields Superior Performance. The combination of the tool selection enhancements from CoTx with the exemplary instances provided by ICL results in optimal performance.

5.2 Error Analysis

We further evaluate and quantify the errors produced by the models, focusing on two main aspects: the occurrence of hallucinations during tool invocation and the model's instruction-following capabilities.

- Hallucinations refer to instances where the model erroneously calls non-existent tools or inputs invalid parameters into existing tools. We denote the set of samples containing hallucinations as $Hall$ and the degree of hallucination in the model is quantified using the metric $P_{Hall} = \frac{Hall}{N_{S1+S4}}$.

- The instruction-following capability primarily assesses the model's ability to generate tool calls in accordance with the specified format. To evaluate this, we employ regular expressions to verify the conformity of tool invocation to the formats outlined in the instructions. We let $From$ represent the number of samples that accurately parse the tool calls, and we utilize the value $P_{From} = \frac{Form}{N_{S1+S4}}$ to gauge the model's proficiency in following instructions.

Model	Methods	S1		S4	
		$P_{Hall}\downarrow$	$P_{Form}\uparrow$	$P_{Hall}\downarrow$	$P_{Form}\uparrow$
GPT-4	zero-shot	26.36	99.12	26.98	98.92
	ICL(3-shot)	25.53	99.84	22.56	99.85
	CoTx	25.45	98.93	23.78	98.97
	CoTx+ICL	23.34	99.29	23.57	99.13
ToolQwen	zero-shot	15.54	98.35	11.74	98.67
	ICL(3-shot)	9.76	99.54	10.03	99.12
	CoTx	12.32	98.83	11.25	98.74
	CoTx+ICL	10.19	99.37	9.45	99.28

Table 3: Error Analysis. A lower P_{Hall} represents lower model hallucination, while a higher P_{From} indicates stronger instruction-following capability of the model.

We also present some examples of errors in Appendix C to facilitate understanding. The experimental results in Table 3 reveal the following points:

- ICL reduces hallucinations and enhances instruction-following capability, while CoTx leads to instability of results. One possible reason is that when the model associates relevant factors, it may create tools on its own to fit the inferred factors without selecting from the specified API list. This situation is significantly alleviated when some samples (CoTx + samples) are provided.

- The model after SFT shows a significant reduction in hallucinations and a noticeably stronger instruction-following ability. This again emphasizes the importance of fine-tuning the model for specific tasks, especially in highly structured and standardized API interaction scenarios.

6 Related Work

Tool Augmented LLMs Equipping large models with interfaces for acquiring external knowledge has become a hot topic in recent years (Gao et al., 2023). Research has covered a variety of tools,

such as code retrieval and execution (Shen et al., 2024; Wang et al., 2023b; Gou et al., 2023), interacting with the world to complete tasks (Wang et al., 2023a; Singh et al., 2023), and retrieving information to answer questions (Zhuang et al., 2023; Parisi et al., 2022; Chen et al., 2022), among others. Tool collection involves open-source available APIs (Qin et al., 2023; Wang et al., 2024) or domain-specific tools developed by engineers (Li et al., 2023). The methods for enabling LLMs to utilize tools can be categorized into two types: (1) Using prompt engineering to help closed-source LLMs understand tool documentation and output the results of tool calls in a specified format. Notable works include (Paranjape et al., 2023; Hsieh et al., 2023; Lu et al., 2024; Gupta and Kembhavi, 2023), where integrated methods like ICL and ReAct (Yao et al., 2022) can seamlessly enhance the model's tool invocation capabilities. (2) For open-source models, knowledge about tools is injected into the LLM through instruction tuning, which further improves the model's performance in tool invocation and adherence to instructions. Representative works include (Tang et al., 2023; Qin et al., 2023; Qiao et al., 2023). However, most works focus solely on improving tool invocation abilities, while significantly fewer studies address the improvement of tool-use awareness (Gui et al., 2024). Our work starts from user intent, comprehensively enhancing the model's decision-making ability regarding the necessity of tool invocation, accurately identifying invocation intentions, and making correct tool calls to better serve users.

Prompting LLMs for Reasoning Advanced prompting methods can significantly enhance the reasoning capabilities of models. Some works, such as CoT (Wei et al., 2022), ToT (Yao et al., 2024), and GoT (Besta et al., 2024), organize the model's gradual thinking in a structured manner to find optimal reasoning results. Other approaches, like ReAct (Yao et al., 2022), empower the model to interact with the world through a think-action-observe cycle, using feedback to make further decisions. Reflection (Shinn et al., 2024) builds on ReAct by adding a reflection mechanism, allowing the model to learn from erroneous attempts. Our CoTx method expands on CoT, clarifying the content of LLM's thinking and reasoning steps. CoTx resembles a binary thinking tree, enabling LLMs to make better decisions and identify implicit intentions in diverse tool invocation scenarios.

Evaluation of Tool Use The automated evaluation of a model's tool usage encounters significant challenges, including the variability of tool invocation results over time, the non-uniqueness of input parameters, and the diverse combinations of tools required to solve specific problems. Currently, the evaluation of tool use encompasses the following approaches: (1) Quality Assessment of Model Response: ToolLLM (Qin et al., 2023) evaluates the quality of the model's tool calls by retrieving real API call results and assessing the pass rate and win rate of the model's responses using GPT-4. To address the timeliness of tool calls, StableToolBench (Guo et al., 2024) achieves stable evaluation results by caching call outcomes and retrieving matches. However, this method's reliability is contingent on the capabilities of advanced language models. (2) String Parsing of Tool Calls: Representative works such as STE (Wang et al., 2024), UltraTool (Huang et al., 2024), and APIBank (Li et al., 2023) evaluate tool calls by parsing them as strings and employing regular expression matching. While efficient for acquiring results, this approach lacks actual tool execution and relies on complex logic for evaluation quality. Given that our work primarily aims to enhance the model's awareness of tool usage and intent recognition capabilities, we adopt the second approach for our assessments. This choice allows us to comprehensively capture the model's performance while ensuring the stability and reliability of our evaluations.

7 Conclusion

In this paper, we introduce PracTool, a user intent-driven tool invocation framework. It encompasses a diverse range of user queries from the real world and aims to comprehensively evaluate a model's tool usage capabilities from three aspects: decision-making regarding tool usage, intent recognition, and tool selection and invocation. We also propose the CoTx reasoning framework, designed to facilitate more thorough thinking and tool usage by the model in diverse scenarios. Extensive experiments demonstrate that PracTool presents significant challenges for advanced LLMs, while the ToolQwen model, fine-tuned based on PracTool, exhibits comprehensive capabilities. Furthermore, the CoTx reasoning framework helps LLMs achieve better tool-use performance. We hope that this research can provide new insights for the study of tool-enhanced LLMs and further promote practical applications.

Limitation

In this work, we propose a user intent-driven tool usage framework that encompasses scenario definition, data generation, model training and reasoning, as well as comprehensive evaluation. We place a strong emphasis on enhancing the model's intent recognition and tool-use capabilities. However, several limitations exist at this stage: (1) The tools utilized in our framework are generated and cannot be executed in real-world scenarios, and their quantity is relatively limited. Given the diversity inherent in the data generation process across various scenarios—especially concerning implicit intent—this restricts applicability to all potential tools. Consequently, we have controlled the types and quantities of tools used to ensure the production of a high-quality dataset. (2) Currently, our data focuses exclusively on the invocation of a single tool. In future work, we plan to broaden the range of tools and extend scenarios to include multi-tool collaborative usage, which will pose additional challenges and enhance the significance of our research.

Ethics Statement

All data in PracTool are de-identified and safeguarding privacy concerns. Our data construction and human evaluation processes are conducted by proficient experts, comprising postgraduate students from Chinese universities and employees of Chinese technology companies, all of whom receive fair payment for their contributions.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2024. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Anchun Gui, Jian Li, Yong Dai, Nan Du, and Han Xiao. 2024. Look before you leap: Towards decision-aware and generalizable tool-usage for large language models. *arXiv preprint arXiv:2402.16696*.

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. *Preprint*, arXiv:2403.07714.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

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*.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, et al. 2024. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *arXiv preprint arXiv:2401.17167*.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.

Marek Kadlčík, Michal Štefánik, Ondřej Sotolář, and Vlastimil Martinek. 2023. Calc-x and calcformers: Empowering arithmetical chain-of-thought through interaction with symbolic systems. *arXiv preprint arXiv:2305.15017*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

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

Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. 2023. Making language models better tool learners with execution feedback. *arXiv preprint arXiv:2305.13068*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

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

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. Llms in the imaginarium: tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. 2023. *Comment: Project website and open-source codebase: https://voyager. minedojo. org/Cited on*, page 33.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023b. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey (2023). *URL https://arxiv. org/abs/2309.07864*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving

with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

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

Hao Zhu Daniel Fried Graham Neubig Zhiruo Wang, Zhoujun Cheng. 2024. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.

A Dataset statistics

We provide the sample size statistics for the four scenarios as shown in Table 4. The data for S2 is obtained by replacing the candidate tool list of the S1 samples.

Scenario	#Samples	Description
S1	3190	Queries with clear intent
S2	1447	Queries without suitable tools
S3	1017	Tool-free Queries
S4	1335	Implicit intent queries

Table 4: Dataset statistics

We also list some sample data in Table 5. Note that the queries for S2 is sampled from S1, and their difference lies in the different candidate tool lists provided to the model. Therefore, in Table 5, we only display the data samples from S1.

B Experimental Details

Implementation Details. In our experiments, we employ the transformers library from HuggingFace to fine-tune our models, where the AdamW optimizer with $\beta = (0.9, 0.999), \theta = 10^{-8}$ are used in the training process. We fine-tune the model for three epoches with learning rate 2×10^{-5} and batch size 16. Meanwhile, we deploy the LoRA-style parameterefficient fine-tuning by using HuggingFace PEFT library, with $lora_r = 256$ and $lora_alpha = 128$. In addition, we also leverage Flash Attention (Dao et al., 2022) and DeepSpeed technologies to reduce the requirement of GPU memory in our experiments. Note that all experiments are conducted on 4 NVIDIA A100 GPUs (80G).

C Error Examples

We present two types of errors made by GPT-4 in Tables 6 and 7: the provision of invalid parameters in API calls, leading to hallucinations, and failure to adhere strictly to the specified output format in the tool calls.

D Prompts

In Table 8, 9 and 10, we provide the prompts for generating clear intent queries, tool-free queries, and queries with implicit intent respectively.

E API List

At the end of the article, we include a list of APIs, detailing their names and parameters.

S1
Query: I want to set a reminder to call my mom and check on her at 9 AM this Mother's Day.
API call: AddReminder(content='Call Mom to check on her', time='2024-05-12 9:00:00')
Query: Our department's small AI project is expected to require the purchase of 10 servers, each costing approximately 12,000 yuan. We need to quickly calculate the total budget to submit for financial approval.
API call: Calculator(formula='10*12000')
Query: I need to confirm whether the BA123 flight scheduled for tomorrow morning at 8 AM will operate as planned.
API call: FlightStatusQuery(flight_number='BA123', date='2024-04-16')

S3
Query: Write a piece of Python code to scrape the highest and lowest prices of **Tesla's stock** on 2022-5-23.
API call: No api call
Query: What are the key points of the currency reforms for the **Turkish lira** and the **Hungarian forint**?
API call: No api call
Query: What role has **gold** played in the history of currency?
API call: No api call

S4
Query: I would like to know if flight ZH9999 will arrive on time on Christmas Day so that my friend can join us for dinner.
API call: FlightStatusQuery(flight_number='ZH9999', date='2024-12-24')
Query: Considering the market fluctuations, is it a wiser choice to invest in gold now compared to last month?
API call: GetGoldPrice()
Query: Do we need to prepare sun protection measures for an outdoor event in Washington, on August 8, 2024?
API call: GetWeather(location='Washington', date='2024-08-08')

Table 5: Data samples for three scenarios. We have bolded the high-frequency keywords in S3 that may trigger tool calls.

User Query: What is the recent trend of the Shanghai Composite Index?
Ground truth API call: QueryStock(index_name = 'Shanghai Composite Index')
Predicted API: QueryStock(index_name='Shanghai Composite Index', market_name = 'Shanghai Stock Exchange' , date = '2024-05-12 15:00:00')

Table 6: Error example of GPT-4 (hallucination). The model provided parameters in the API definition that do not exist, resulting in hallucinations.

User Query: What is the recent trend of the Shanghai Composite Index?
System Prompt:... Return the tool call result in the format [ApiName(key1=..., key2=...)]...
Expected Output: [QueryStock(index_name = 'Shanghai Composite Index')]
Model Output 1: [' QueryStock(index_name = 'Shanghai Composite Index') ']
Model Output 2: QueryStock(index_name = 'Shanghai Composite Index')

Table 7: Error example of GPT-4 (instruction-following). The model's output format has slight differences from the format requested in the prompt, which leads to difficulties in parsing and results in errors in instruction following.

You are an assistant skilled in utilizing various tools in different ways. Now, I will provide you with a tool, along with some queries and examples of using these tools:
Tool: {tool}
Examples:{samples}
At the same time, I will give you a character profile: {profile}
You need to embody this character and fully understand their age, personality, background, etc. Additionally, your task is to reference the above examples of tool usage and generate as many diverse user queries as possible from a first-person perspective, along with the corresponding API calls. When the queries involve time, you need to provide a specific time, clearly indicating the year, month, day, and exact moment. For example, 2024-5-6 12:24:00, rather than vague time references like "every Friday, next Wednesday, tomorrow," etc. This year is 2024.
==========
Expected output:
"User": your query
"API": [Apiname(key1=..., key2=...)]

Table 8: Prompt for generating queries with clear intents.

You are an assistant skilled in utilizing various tools in different ways. Now, I will give you a tool:
Tool: {tool}
and the high-frequency keywords related to using this tool:
Keywords: High-frequency keywords
You need to have a comprehensive understanding of the tool's definition and function. Your task is to generate as many diverse user queries as possible from a first-person perspective, which should be related to {tool} and include the keywords High-frequency keywords, but unrelated to actually calling the tool. For example:
{input_examples}.
Additionally, when the queries involve time, you need to provide a specific time, clearly indicating the year, month, day, and exact moment. For example, 2024-5-6 12:24:00, rather than vague time references like "every Friday, next Wednesday, tomorrow," etc. This year is 2024.
==========
Expected output: "User": your query

Table 9: Prompt for generating tool-free queries.

You are a help assistant in asking questions. I will provide you with an initial question, the intent corresponding to that initial question, and the tool API that will be called to address the initial question, for example:
{pre_context}
Your task is to propose a follow-up question based on the initial question. Both questions should have the same intent, exhibit semantic continuity, and invoke the same tool, for example:
{follow-up question examples}
The initial question and follow-up question in the above example have the same intent and semantic coherence, making them likely to appear in a continuous conversation. Additionally, you should adhere to the following rules:
1) Note: Your generated follow-up question must not contain overly obvious intent phrases like {explicit_intents}.
2) The parameters of the follow-up question when calling the tool must remain consistent with those of the initial question.
3) You must ensure the generated follow-up question is diverse.
4) Avoid vague time references, such as [that day, recently, within the next six month]. If the initial question includes a specific time, your follow-up question should repeat that time.
5) Now, please fully understand the examples above and generate one follow-up question for the following initial question:
{input_question}
==========
Follow-up question:

Table 10: Prompt for generating queries with implicit intents.

```
API List:
{
    "type": "function",
    "function": {
        "name": "AddReminder",
        "description": "This API is used to add a reminder, including
            content and time.",
        "parameters": {
            "type": "object",
            "properties": {
                "content": {
                    "type": "string",
                    "description": "The content of the reminder."
                },
                "time": {
                    "type": "string",
                    "description": "The time for the reminder. Format
                       : %Y-%m-%d %H:%M:%S"
                }
            },
        }
    }
},
```

```
{
    "type": "function",
    "function": {
        "name": "Calculator",
        "description": "This API provides basic arithmetic operations
           : addition, subtraction, multiplication, and division.",
        "parameters": {
            "type": "object",
            "properties": {
                "formula": {
                    "type": "string",
                    "description": "The formula to be calculated.
                       Only integer values are supported. Valid
                       operators are +, -, *, /, and ( ). For example
                       , '(1 + 2) * 3'."
                }
            },
        }
    }
},
{
    "type": "function",
    "function": {
        "name": "PlayMusic",
        "description": "This API triggers the music player to play
           music.",
        "parameters": {
            "type": "object",
            "properties": {
                "music_name": {
                    "type": "string",
                    "description": "The name of the music to be
                       played."
                }
            },
        }
    }
},
{
    "type": "function",
    "function": {
        "name": "Translate",
        "description": "Translate text into the target language.",
        "parameters": {
            "type": "object",
            "properties": {
                "src": {
                    "type": "string",
                    "description": "The text to be translated."
                },
                "src_lang": {
                    "type": "string",
```

```
                        "description": "[Optional] Source language.
                           Default is automatic detection."
                    },
                    "tgt_lang": {
                        "type": "string",
                        "description": "[Optional] Target language.
                           Default is English (english/en)."
                    }
                },
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "GetWeather",
            "description": "Query weather conditions: temperature,
               humidity, AQI, weather, wind direction, etc.",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The location to query, as
                           detailed as possible, for example: `Haidian
                           District, Beijing` or `Minhang District,
                           Shanghai`."
                    },
                    "date": {
                        "type": "string",
                        "description": "Date, format: YYYY-MM-DD, and the
                            date must be less than today."
                    }
                },
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "QueryStock",
            "description": "Get stock prices based on company name or
               stock code.",
            "parameters": {
                "type": "object",
                "properties": {
                    "market_name": {
                        "type": "string",
                        "description": "Market name."
                    },
                    "index_name": {
                        "type": "string",
```

```
                    "description": "Index name."
                },
                "company_name": {
                    "type": "string",
                    "description": "Company name."
                },
                "code": {
                    "type": "string",
                    "description": "Stock code."
                }
            }
        },
    }
},
{
    "type": "function",
    "function": {
        "name": "GetExchangeRate",
        "description": "Query the currency exchange rate based on
           currency codes.",
        "parameters": {
            "type": "object",
            "properties": {
                "source": {
                    "type": "string",
                    "description": "Source currency code."
                },
                "target": {
                    "type": "string",
                    "description": "Target currency code."
                }
            },
        }
    }
},
{
    "type": "function",
    "function": {
        "name": "GetGoldPrice",
        "description": "Query the gold price.",
        "parameters": {
            "type": "object",
            "properties": {}
        },
    }
},
{
    "type": "function",
    "function": {
        "name": "GetSilverPrice",
        "description": "Query the silver price.",
        "parameters": {
```

```
                "type": "object",
                "properties": {}
            },
        }
    },
    {
        "type": "function",
        "function": {
            "name": "GetOilPrice",
            "description": "Query today's domestic oil prices.",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The location to query, as
                           detailed as possible, for example: `Haidian
                           District, Beijing` or `Minhang District,
                           Shanghai`."
                    }
                },
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "GetCityTime",
            "description": "Query the current time of a specific city.",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {
                        "type": "string",
                        "description": "The name of the city to query the
                            time for, for example: `New York`."
                    }
                }
            },
        }
    },
    {
        "type": "function",
        "function": {
            "name": "FlightStatusQuery",
            "description": "Query flight status.",
            "parameters": {
                "type": "object",
                "properties": {
                    "flight_number": {
                        "type": "string",
```

```
                    "description": "Flight number, for example: `
                        CA1234`."
                },
                "date": {
                    "type": "string",
                    "description": "The date to query, for example:
                        `2023-10-01`."
                }
            },
        }
    }
},
{
    "type": "function",
    "function": {
        "name": "CalculateBmi",
        "description": "Calculate Body Mass Index (BMI).",
        "parameters": {
            "type": "object",
            "properties": {
                "weight": {
                    "type": "number",
                    "description": "Weight, in kilograms."
                },
                "height": {
                    "type": "number",
                    "description": "Height, in meters."
                }
            }
        },
    }
},
{
    "type": "function",
    "function": {
        "name": "RecipeLookup",
        "description": "Query cooking recipes.",
        "parameters": {
            "type": "object",
            "properties": {
                "dish_name": {
                    "type": "string",
                    "description": "Name of the dish."
                }
            },
        }
    }
},
{
    "type": "function",
    "function": {
        "name": "HealthGuideline",
```

```
            "description": "Provide health guidelines and advice.",
            "parameters": {
                "type": "object",
                "properties": {
                    "condition": {
                        "type": "string",
                        "description": "Health condition, for example: `
                           diabetes`, `heart_disease`."
                    }
                },
                "required": ["condition"]
            }
        }
    }
```