

Benchmark	Tool Learning						Evaluation			
	Apps	APIs	Argu.	C. S. E.	States	Awareness	Role	Hierarchical	Resp.	Multi-turn
APIBench (Patil et al., 2023)	3	1,715	(1.5/1.0)	X ✓ ✓	X	X	X	X	X	X
API-Bank (Li et al., 2023)	8	53	(2.5/1.0)	X ✓ ✓	X	X	X	X	✓	✓
ToolBench (Qin et al., 2023c)	49	16,464	(1.0/1.0)	X ✓ ✓	X	X	X	✓	X	✓
ToolQA (Zhuang et al., 2023)	6	13	(1.0/1.0)	X ✓ ✓	X	X	X	X	✓	X
GAIA (Mialon et al., 2023)	-	-	-	X ✓ ✓	X	X	X	X	✓	X
UltraTool (Huang et al., 2024)	22	2032	(4.1/1.6)	✓ ✓ ✓	X	X	X	X	X	X
AgentBench (Liu et al., 2023)	8	-	-	X ✓ ✓	X	X	X	X	X	X
MINT (Wang et al., 2024c)	8	-	-	X ✓ ✓	X	X	X	X	✓	X
AgentBoard (Ma et al., 2024)	9	-	-	X ✓ ✓	✓	X	X	X	X	✓
DialogTool	16	31 [♡]	(4.2/7.5)	✓ ✓ ✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison with existing evaluation benchmarks (first part: tool learning benchmarks; second part: agent benchmarks) where the C.S.E. stands for Creation | Selection | Execution of tools, and ✓ stands for the selection of tool does not consider the case which does not need any tools. ♡ We emphasize that we focus on the interaction and complexity of each API instead of solely number of APIs. Thus we list the average number of input and returned arguments by APIs in Argu. Culumn. Hierarchical stands for hierarchical tool design in our VirtualMobile in terms of App, API and Arguments.

To maintain seamless interaction over long horizons, we introduce DialogTool, the first benchmark designed to comprehensively evaluate the entire lifecycle of stateful tool use in multi-turn dialogues. Table 1 compares DialogTool with existing benchmarks. Generally, we leverage existing dialogue datasets, particularly task-oriented dialogue datasets (TDD) (Budzianowski et al., 2018; Ras-togi et al., 2020), to gather data and construct the corresponding evaluation environment efficiently and effectively. In detail, on the data side, we regard the *service/domain*, *slots* and *intents* in TDD as different *Apps*, *Arguments*, and *APIs*, and transform every database lookup operation in the dialogue into an API function call, adhering to the standard tool call paradigm (Li et al., 2023; Wang et al., 2024b). On the environment side, we firstly store the output for each API call as the database, and then manually implement each function for all APIs and ensure the correctness², resulting in a virtual mobile environment (VirtualMobile) with lots of supported Apps and APIs. For example, the user may want to find one restaurant with specific food and location, and the result can be returned using FindRestaurant API in Restaurant App that takes the desired food type and location as input parameters, and returns a list of names of matching restaurant.

Building on top of DialogTool and VirtualMobile, we can assess the entire lifecycle of stateful tool use by examining six dimensions across three different stages (Figure 1): 1) **Tool Creation** to generate code function given

²Given same input in the dialogue, it can produce same output,

the whole tool description; 2) **Tool Utilization** which consists of tool awareness to determine whether or not require tools, tool selection to select appropriate API and tool execution to fulfill all arguments; and 3) **Role-consistent Response** to generate final responses according to different roles (i.e., role play) and tool states (i.e., response generation). It is worth noting here that the role playing transforms responses into different styles to enhance user engagement, independent of the tools being used, allowing for varied expressions regardless of the specific tools employed. To conclude, our contributions can be summarized below:

- To the best of our knowledge, this is the first attempt to evaluate the whole life cycle of stateful tool use in the context of multi-turn dialogue, including six key dimensions across three distinct stages.
- We propose DialogTool, the first multi-turn dialogue dataset considering stateful and interactive tool use, accompanying with an embodied virtual mobile environment (VirtualMobile), ensuring the reproducibility and evaluation of different LLMs to interact with both humans and APIs over long horizons.
- We conduct extensive experiments on 13 distinct LLMs of varying sizes, covering both state-of-the-art open- and closed-source models, and then provide comprehensive analysis in each stage of tool use and address the challenges encountered in practice.

2 Related Work

Task-oriented Dialogue. Task-oriented Dialogue Systems (DS) have undergone significant transformations with the progress of Language Models (LMs) (Wang et al., 2023b). Despite the differences in models, the core of a dialogue system lies in determining the next action and coordinating various knowledge from different services to complete the task. For example, Rastogi et al. (2020) propose a Schema-Guided Dialogue (SGD) dataset considering an ever-increasing number of services and APIs spanning multiple domains. However, the majority of current dialogue systems lean towards traditional frameworks while utilizing LLMs as their foundational models (Hudeček and Dusek, 2023; Zhang et al., 2023), benefiting from well-established techniques. In contrast, several advancements have begun to explore tool learning in dialogue system, treating different APIs (Shu et al., 2022; Li et al., 2023) or knowledge sources (Wang et al., 2023a; Pan et al., 2023) as individual tools. Building on this, recent studies have increasingly focused on complex tool interactions under specific constraints, such as domain policies (Yao et al., 2024), drawing inspiration from traditional task-oriented dialogue research (Xu et al., 2024; Lu et al., 2024).

Tool Learning. Tools have been defined as cognitive tools or physical tools (Wang et al., 2024a), where the former is defined as a cognitive concept inside human beings comes from cognitive science (Gigerenzer, 1991; Baron-Cohen, 1991; Wang et al., 2023c) and the later comes from external physical world such as different models (Shen et al., 2023), search engine (Wang et al., 2025b), retrievers (Shen et al., 2025), APIs (Li et al., 2023; Wang et al., 2024b), and even robot manipulation (Huang et al., 2022; Liang et al., 2023). Most of previous works, such as APIBench (Patil et al., 2023) and ToolQA (Zhuang et al., 2023), have primarily revolved around the selection and execution of tools. This includes tasks such as identifying the right tool for a given instruction and understanding all the necessary arguments needed to execute the determined tool. Furthermore, Mialon et al. (2023) consider the cases which do not require tools and require multiple tools in single turn. AgentBoard (Ma et al., 2024) consider the progress evaluation at each step to complete the complex task. Moreover, recent studies start to explore more efficient and effective tool utilization via reinforcement learn-

ing (Wang et al., 2025a; Qian et al., 2025). He et al. (2025) propose a novel training framework that prepares LLMs for diverse generalization challenges in tool utilization. Distinguishing from these previous works, we focus on the whole life cycle of tool utilization across three distinct stages, and introduce more fascinating features such as role playing and practical hierarchical structure³.

Role Play. Assign LLMs some pre-defined roles has been proved an effective way to engaging user (Wang et al., 2023d), resulting in more longer interaction time, such as character.ai⁴. Most existing work focuses on character roles rather than assistant roles. For example, CharacterEval (Tu et al., 2024) evaluates LLMs on generating role-consistent responses based on a given role background. In contrast, we focus on the language style of different roles (Zhou et al., 2023), aiming to generate more user-friendly and preferable responses.

3 Dataset and Environment

3.1 Seed Dataset

To create our DialogTool dataset effectively and efficiently, we prioritize using existing task-oriented dialogue datasets (TDD) that closely resemble real-world interactions while minimizing human effort. We first select seed datasets based on two main criteria: 1) The datasets should well reflect how tools or functions are invoked as the conversation goes, such as dialogue system call different APIs in the multi-turn task-oriented dialogue dataset; 2) We prefer datasets that offer diverse conversations with comprehensive and detailed annotations, ranging from different domains and tools. Specifically, we mainly incorporate the SGD dataset (Rastogi et al., 2020) and also Multi-WoZ (Budzianowski et al., 2018; Zang et al., 2020) due to their extensive coverage across various domains, slots, and slot values, featuring a dynamic ontology of APIs spanning numerous domains.

3.2 Dataset Collection

Figure 2 shows the details of data collection. In detail, there are two steps: 1) *Setting Shift*: we write a Python script to automatically transform the annotations of task-oriented dialogue dataset (except the utterances), into conventional labels in tool learning (i.e., tool selection and execution); 2)

³More related work can be found in Appendix.

⁴<https://character.ai/>

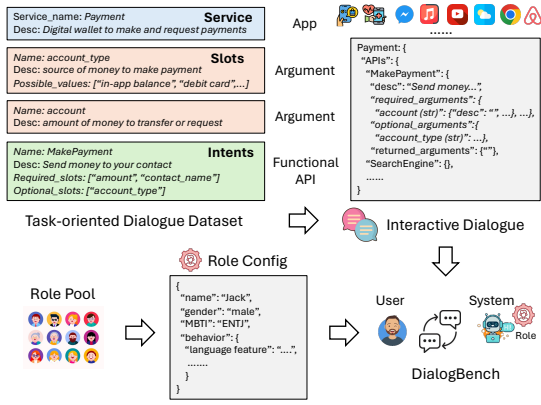


Figure 2: The pipeline of DialogTool collection by 1) **Setting shift**: transfer the setting of existing dialogue datasets; 2) **Role Play**: then rewriting the utterances from system side into role-consistent utterances.

Role Playing: In order to provide role-consistent response, we assign dialogue agent different roles, resulting in varied response styles.

Setting Shift. Lots of recent studies try to build tool learning benchmark from scratch (Li et al., 2023; Yao et al., 2024), being time-consuming and labor-intensive. Alternatively, existing task-oriented dialogues share lots of same assumption and similarities with tool learning interactions, such the concept of domain (or services), intent, slots and corresponding values to help the users to complete the predefined tasks (Budzianowski et al., 2018; Zhu et al., 2020). Therefore, it is natural and straightforward to re-formulate task-oriented dialogues as a multi-turn interactions with tool and language feedback. To achieve this, we make several essential adjustments to align with the requirements of tool learning across the action, App, API and argument levels.

- **Actions**. The actions in TDD are typically defined in the format of *intent-domain-slot-value* in lots of previous works (Kale and Rastogi, 2020; Kwan et al., 2024). For example, "request-restaurant-name-?" indicates that the system needs to request the name of the restaurant from the user. We save the first slot (a.k.a., request) since the other keys are all related to external tools/services, and then categorise them into *non-tool* actions and *tool-related* actions (shown in Table 7), considering lots of dialogue turns do not require tool calls. – **Tool Awareness**⁵

⁵We note that our tool awareness is not simply binary classification (yes or no) but more fine-grained classification based on tool states.

- **Apps**. We consider different domains/services as different Apps in a virtual mobile phone. Specifically, the schema of each App contains three key fields besides the API functions: 1) *description*: which describes all tasks supported by the App in natural language; 2) *base required arguments*: which provides information about base arguments required by all supported API in the App, such as user name and password; 3) *APIs description*: using the name of API as key and store all necessary information about the APIs (see below). This kind of design enables more flexible and easy implementation by simply passing different fields to LLMs to decide which App or API to call. – **Tool Selection**

- **APIs**. Following (Rastogi et al., 2020), each API includes the *name*, *description*, a *flag*⁶ which indicates that the underlying API call is transactional (e.g, a booking or a purchase) as opposed to a search call, along with *additional required* and *optional arguments*. This setup closely resembles a function call in a programming language, particularly with regards to optional arguments. For instance, when renting a car, individuals may have varying preferences for the type of car they prefer. Taking these preferences into account enhances the tool’s learning by making it more personalized and customized. The API can only be executed when all base required arguments and all additional required arguments are filled. – **Tool Selection**

- **Arguments**. Each argument is defined with the format: "*name (type)*": "*description*". For example, the argument *start_date* in the 'getcarsavailable' API is defined as "*start_date (date)*": "the first date to start using the rental car, the format follows yyyy-mm-dd". We emphasize that the format of arguments is crucial when calling APIs, as they often expect specific structures or data types to function correctly. This is where LLMs can be particularly useful to interpret natural language instructions and convert them into the precise format required by the API. Using the same example above, the human may provides start date in a more casual format like "tomorrow" or "next Monday", LLMs need to translate that into the appropriate format (yyyy-mm-dd) for the API to understand (a.k.a, *arguments formatting*). An example can be found in Fig 5. –

⁶This is unique in SGD datasets since it requires the system to confirm before call the transactional API.

Tool Execution

Role Playing. In order to provide customized experience and engaging users, we manually gather 50 roles (from movies or TV shows) along with their respective configurations, which include *name*, *gender*, *MBTI type*, and *behavioral attributes* following recent studies (Zhou et al., 2023). These attributes encompass *language features*, *emotional expressions*, and *interaction patterns*, leading to different conversation styles of dialogue assistant. Since all the previous operations do not change the content of utterances of the user and system except the progress annotations of each turn in multi-turn dialogues, we can directly prompt different LLMs to convert original system responses⁷ into role-consistent expression. This allows us to mimic how different individuals might convey the same results returned by APIs in distinct ways since response styles are orthogonal to the API results.

3.3 Environment Set Up

To mirror real-world agent-tool interactions, we need to carefully construct the tool environment and collect corresponding database. Firstly, we manually implement each App and API in python language by using the name of App as class name and each API within App as one function. We additionally add language descriptions for each App, API and corresponding arguments, and store them as special attribute of App class. There are other necessary functions and attributes to track the states of different App and APIs as the conversation goes on. Secondly, we sample every database lookup operation from the original dialogue datasets and store all unique returned results as the database for each App. For example, if the restaurant lookup operation returns several different candidates, we can store them together into the database. Afterwards, we can successfully build an virtual mobile environment – VirtualMobile which enables interactive tool utilization and validation of correctness of created tools.

3.4 Data Analysis

Table 2 presents the data statistics of our proposed DialogTool. The whole dataset contains 20 different Apps and 45 distinct APIs⁸. Most Apps contain at least 2 APIs and 5 at maximum, covering lots of user needs in practice, such as booking

⁷We do not need to change user’s utterance, and we find this does not affect the natural flow of the dialogues.

⁸The full list of Apps and APIs can be found in Appendix.

Statistics	Training	Evaluation
# of Apps	20	15
# of APIs	45	30
# of Dialogues	16,142	900
# of Multiple Apps	10,739	360
# of Turns	329,964	15,568
# of Calls	85,191	4,274
# of Roles	50	16
Avg. turns	20.4	17.3
Max. input arguments	9	7
Avg. input arguments	3.3	3.8

Table 2: The data statistics of DialogTool. # multiple apps means the number of dialogues where multiple apps are used.

a hotel/restaurant/flight, renting a car, finding near events and others. In general, DialogTool comprises approximately 16k dialogues and 33k turns, surpassing existing benchmarks like API-Bank (Li et al., 2023) by a significant magnitude. This scale enables comprehensive exploration of dialogue system capabilities across a vast corpus of interactions. Moreover, the dataset’s average of around 4 input arguments per API during evaluation and over 16 turns per dialogue highlights the depth and complexity inherent in user-dialogue system interactions. This complexity is further underscored by the prevalence of multi-App dialogues, accounting for 50% during training and 36% during evaluation, showcasing the real-world challenge of orchestrating seamless interactions across multiple APIs.

4 Experiments

4.1 Task Definition

Given the dialogue context $c = \{u_1, s_1, \dots, u_t\}$ and a virtual mobile environment $\mathcal{E} = \{App_1, App_2, \dots, App_n\}$ where each App contains several APIs $\{p_i^1, \dots, p_i^j\}$, and corresponding environmental state e_{t-1} at current turn t , the dialogue agent either interact with the environment and then generate the final dialogue response s_t according to updated state e_t and previous context, or directly generate the response s_t based on current state and context since it is not required to call the API in the environment.

4.2 Set Up

Considering the complexity and interactivity of the whole life cycle of stateful tool use, we present more details about the evaluation for each dimension.

Tool Creation. To assess LLMs’ ability to develop new APIs, we provide them with complete

information about the API, including its description, and all arguments (i.e., required and returned arguments) alongside with one demonstration. This ensures that the LLM understands the input, output, and purpose of the API before generating corresponding functions. Then we utilize all API calls in the test dialogues as test cases to evaluate whether or not created API (in the python function format) can successfully return the same results given same input arguments⁹. Furthermore, this strict evaluation helps minimize the impact of code hallucination. We emphasize that this can be achieved on the fly during the conversation if existing toolsets are insufficient.

Tool Awareness. Previous studies simply consider tool awareness as a binary classification problem such as using or does not use tools. However, it becomes inadequate in the context of stateful tool use since a user may inquiry about results of previous tool calls without necessitating a new tool invocation. Therefore, a more nuanced evaluation is needed to accurately reflect the complexities of stateful interactions. We consider more practical actions which support stateful multi-turn interactions as shown in Table 7, and we prompt the dialogue agent to select correct action a from the given list based on current dialogue context c .

Tool Selection and Execution. Once the previous determined action necessitates the API call, the dialogue agent needs to select the most appropriate API from the whole API list supported in the environments given the dialogue context c and the environment $\mathcal{E}$, following the format of $\{t(k_1 = v_1, \dots, k_m = v_m)\}$. The k and v stands for the name and value of each argument of the API. We also consider hierarchical selection strategy which select appropriate App first and then select appropriate API, in order to better reflect the interactions of real-world applications. After it try to execute the tool at the environment, the environmental state will be updated to e_t and then be used to generate the response. We calculate the accuracy at the API level and Argument level¹⁰ respectively.

Response Generation. According to the determined action, the dialogue agent generate the final

response s_t based on context c and environmental state e , such as requiring more details about arguments and providing alternative suggestion regarding previous tool call results.

Role Play. We additionally evaluate whether or not the dialogue agent can play different roles to transform the dialogue response s_t into different styles s_t^r . Specifically, we randomly sample one role from predefined role list for one dialogue, the dialogue agent is tasked with generating role-consistent response.

4.3 Implementation Details

Models. We choose 13 distinct models whose size ranging from 6B to the 72B, aiming to provide comprehensive evaluation for current LLMs, following Wang et al. (2024b). Specifically, we choose ChatGLM (Du et al., 2022) (chatglm3-6b), Qwen series (Bai et al., 2023) (Qwen1.5-7B/14B/72B-Chat), Mistral (Jiang et al., 2023) (Mistral-7B-Instruct-v0.2), LLaMa2 series (Touvron et al., 2023) (Llama-2-7b/13b/70b-chat-hf), and latest LLaMa3 series (AI@Meta, 2024) (Meta-Llama-3-8B/70B-Instruct) for open-source LLMs. Besides that, we also select latest GPT3.5 (gpt-3.5-turbo) and GPT-4o (gpt-4o) from OpenAI for closed-source LLMs. We set temperature and top p as 0.1 to reduce randomness. All experiments are run on NVIDIA A100 GPUs.

Metrics. For tool creation, we focus on the generated function can pass all test cases available in the DialogTool. The pass rates of each API function are then aggregated to determine overall performance (i.e., the ratio of passed test cases to total test cases), aligning with code tests¹¹. For tool utilization, we adopt *accuracy* to evaluate the performance following (Huang et al., 2024). To assess the quality of generated responses, we employ well-established metrics such as BLEU and Rouge.L following previous studies (Li et al., 2023). Furthermore, we employ GPT-4o to evaluate the consistency of roles depicted within the responses (Zhou et al., 2023)¹². We also conduct a human evaluation to validate the alignment of our response evaluation setting with human judgements. We provide all details about the prompts and human study at the Appendix to ensure the reproducibility.

⁹The average (min) test cases for each API is 363 (93), and there are a total of 32 APIs that need to be created.

¹⁰If the user does not specify all necessary arguments in one turn, we let LLMs to replace the value with "?" for these missing arguments.

¹¹<https://leetcode.com/>

¹²We try other models such as Llama3.1-70B-Instruct and we do not observe significant differences.

Models	Tool Creation	Tool Utilization			Role-consistent Responses			
		Awareness	Selection	Execution	BLEU	R.L	Role	Human
ChatGLM3-6B	31.5	58.9	32.8	6.8	7.8	7.5	4.8	1.64
LLaMA2-7B	33.2	63.5	27.4	7.0	6.8	5.7	6.2	1.25
QWen1.5-7B	21.9	<u>68.9</u>	54.7	11.3	8.0	7.4	7.0	<u>2.82</u>
Mistral-7B	11.4	42.5	51.8	22.6	8.0	7.1	6.7	2.28
LLaMA3-8B	<u>62.2</u>	46.3	<u>61.4</u>	<u>45.6</u>	<u>8.3</u>	<u>7.7</u>	<u>7.0</u>	2.69
LLaMA2-13B	<u>48.8</u>	47.1	51.1	11.7	7.7	6.4	6.5	2.17
Vicuna-13B	-	<u>64.5</u>	<u>62.9</u>	12.3	<u>10.1</u>	<u>11.5</u>	6.0	<u>2.59</u>
QWen1.5-14B	27.9	51.7	55.6	<u>21.8</u>	9.3	10.9	<u>7.5</u>	2.44
QWen1.5-72B	49.7	75.5	<u>71.9</u>	<u>49.3</u>	<u>10.8</u>	15.3	7.4	<u>3.37</u>
LLaMA2-70B	23.0	34.7	57.8	32.6	8.5	10.7	6.2	2.56
LLaMA3-70B	69.7	40.2	57.1	<u>68.1</u>	9.0	11.3	<u>7.7</u>	2.98
GPT-3.5	63.3	<u>67.9</u>	50.0	42.6	10.2	11.9	6.7	3.42
GPT-4o	<u>66.7</u>	63.5	77.8	68.7	11.4	<u>14.5</u>	8.3	3.56

Table 3: The main results of DialogTool at three stages: 1) Tool Creation; 2) Tool Utilization; 3) Role-consistent Response. **Bold** highlights the best score and underline underscores the best score under the same model scale.

4.4 Main Results

Table 3 shows the results of whole lifecycle of stateful tool use, several observations can be drawn as follows.

Overall. On the model side, GPT-4o outperforms other models in most cases, while QWen1.5-72B and LLaMA3-70B show competitive performance against GPT-4o. It is observed that the performance correlates positively with model size, particularly within the same model family. On the task side, no LLMs achieve an accuracy exceeding 80% at the tool creation and utilization, and most LLMs performed poorly in tool creation and execution tasks compared to their performance in awareness and selection tasks, revealing the complexity and challenges of our dataset and environment.

Tool Creation. We find that LLaMA3 series model achieves exceptional performance at tool creation, such as LLaMA3-70B outforms GPT-4o and LLaMA3-8B is comparable with GPT-3.5. It can be attributed to additional code pre-training at the LLaMA3 models. In addition, Vicuna-13B can not pass the tool generation task and get no evaluation result, since the generated code has no indent and is not executable by python interpreter. Similar situation is observed on Mistral-7B, however, in some cases, Mistral-7B can still generate executable code, resulting in a low but non-zero pass rate.

Tool Utilization. (1) **Awareness.** It is observed that the performance is not improved consistently as size increases, as validated by both QWen and LLaMa2. In addition, most of LLMs prefer not to

use external tools (i.e., decide more non-tool actions: *inform* or *offer_intent*) no matter small-sized models (Mistral-7B, LLaMA3-8B) or large-sized models (LLaMA2-70B). (2) **Selection and Execution.** A successful tool execution requires the correctness of API and all necessary arguments in the required format. Therefore, we can observe the performance of selection is better than execution in almost all LLMs, revealing the complexity of tool execution.

Role-consistent Response. (1) **Response.** The larger the model size and the more accurate the tool utilization, the better the responses. This is reasonable since the results of tool utilization highly affect the quality of system response and the flow of conversation. For instance, LLaMA3-8B achieves better performance compared with other 7B models, and LLaMA3-70B further boost the performance due to increased size and more accurate tool utilization. (2) **Role Playing.** Larger models generally tend to deliver better performance, despite the gap in role-consistent scores across different LLMs being relatively small. (3) **Human Eva.** The trend is similar with what we observed during main experiments. The larger models tends to lead better performance, while GPT series models achieve best performance compared with other models.

5 Analysis

To offer a comprehensive evaluation of whole lifecycle of stateful tool use, we conduct error analyses for tool creation and tool utilization. In addition, we also explore the effects of different selection

Models	Tool Awareness ($\downarrow$)		Tool Selection ($\downarrow$)		
	T.	No T.	Rec.	API.	Unn.
QWen1.5-7B	2.5	28.6	76.8	0.8	28.9
LLaMA3-8B	20.8	32.9	64.6	1.2	4.0
LLaMA2-13B	19.6	33.3	59.9	4.1	13.1
QWen1.5-14B	21.8	26.5	46.8	11.3	3.7
QWen1.5-72B	7.3	17.2	84.2	0.2	11.5
LLaMA3-70B	39.2	20.6	82.6	0.0	0.4
GPT-3.5	4.8	27.3	68.6	0.1	27.1
GPT-4o	19.9	16.7	91.4	0.9	0.3

Table 4: The error analysis of tool awareness and tool selection. **T.**: the rates of cases that LLMs should use tool but the action does not invoke it (false negative); **No T.**: The rates of cases that LLMs should not use tool but the action invoke tool call or invoke other types of *non-tool* actions (false positive). **Rec.**, **API.** and **Unn.** stand for the recall, API parsing error and unnecessary tool calls.

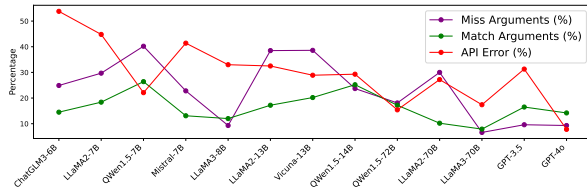


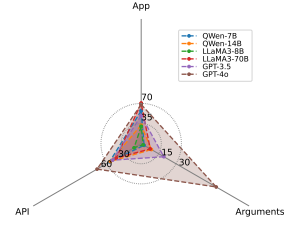
Figure 3: The three primary errors at tool execution.

strategies ¹³.

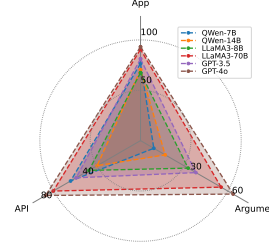
Tool Creation Figure 13 shows the detailed tool creation performance of each API for each LLM. From the results, we can find that: *Model size does not generally contribute to a higher performance.* We attribute this findings to the ‘hallucination’ of code generation in tool creation task. For example, LLaMA2-70B tends to use `sqlite3` library that is not supported and required in the API description, while the smaller LLaMA2-7B and LLaMA2-13B does not have this issue. *Complex tool is harder to be created than simple tool.* We observe a general lower performance across all LLMs on complex APIs like `gettraintickets` in the Train App which has 6 required arguments. In contrast, we find a general better performance on simple APIs like `schedulevisit` in the Home App that only has 2 required arguments.

Tool Utilization i) *Tool Awareness and Selection.* Table 4 shows the error analyses. On the one hand, we can find LLMs struggle to select the timing to use tool, resulting in a high rate of false negatives or false positives. Furthermore, the performance gap across different LLMs mainly comes from the

¹³More analysis can be found in Appendix.



(a) Tool Selection under *Flat* Setting



(b) Tool Selection under *Hierarchical* Setting

Figure 4: The performance of two prompting strategies.

FN part especially for powerful LLMs since they tend to not use external tools. On the other hand, the recall is usually higher than acc regardless of the type of LLM. This indicates that once LLMs know when to call tools, they have a good chance of choosing the correct one. Moreover, we can attribute the relatively lower accuracy of GPT-3.5 to the 27.1% of its tool calls being unnecessary; ii) *Tool execution.* Figure 3 shows the three major errors. It is observed that most of error cases comes from missing arguments instead of arguments matching (i.e, name and value matching). In addition, API error remains a significant challenge, highlighting the difficulties associated with error propagation in the task.

Hierarchical Tools We examine the performance of LLMs in selecting APIs when the App is not known in advance. Therefore, we explore two scenarios: 1) *Flat*: select APP and API together; and 2) *Hierarchical*: select APP first and then API. Figure 4 illustrates the results. *Generally, we can find that models likely achieves much higher performance under hierarchical setting instead of flat one.* We emphasize this not only support the effectiveness of our introduced hierarchical structure, but also more aligns with the cases in practice since it is usually impossible to access all APIs inside each App for the dialogue agent. *Furthermore, it is noteworthy that the performance of LLMs on arguments level is worst compared with App and API selection in both settings,* since it requires LLMs to

Turns	Tool Awareness	Tool Selection	Tool Execution
10	78.9	85.7	75.2
20	59.7	69.5	51.2
30	56.7	60.1	38.5
40	52.3	54.4	35.3

Table 5: Performance across different number of turns for Tool Awareness, Selection, and Execution.

recognize and extract the values for each arguments from multi-turn dialogue interaction, and then format them in the required format. This also aligns with our main experimental results.

Effects of Number of Turns. We additionally conduct performance of number of turns based on GPT-4o model. Specifically, there are many turns in the multi-turn dialogues ranging from 4 to more than 40 turns. Table 5 shows the final results. It is found that as the number of turns increases, the performance drops accordingly, especially for the tool execution.

6 Conclusion

We introduce DialogTool, which is the first multi-turn interactive benchmark considering the whole life cycle of tool utilization of dialogue agents, spanning across three distinct stages, six different tasks. Furthermore, we build VirtualMobile, an virtual mobile evaluation environment to simulate API calls and return corresponding results. We hope these benchmark and environment will provide a comprehensive platform for evaluating and advancing dialogue agents’ tool utilization capabilities, fostering future research in multi-turn interactions.

Limitation

Our study comes with two minor limitations. On the one side, each App contains a limited number of APIs, focused primarily on key functions. We start with the most commonly used Apps and APIs in the real-world and target the interactions between the user, dialogue agent and the environment. As revealed by recent studies (Yao et al., 2024), it is hard for existing advanced LLMs to successfully complete the task over long horizons even considering only two simple situations, such as retail and airline.

On the other side, we do not consider the existing agent framework since we mainly focus on the base capabilities of various LLMs on this new benchmark. We anticipate that introducing additional

reflection or a carefully designed agent workflow may further boost the performance on this setting.

Ethical Considerations

In conducting our research, we have thoroughly reviewed and ensured compliance with ethical standards. Our study utilizes existing datasets, which have been publicly available and previously vetted for ethical use. These datasets have been carefully selected to avoid any form of offensive or biased content. Therefore, we consider that our research does not present any ethical issues. The data used is ethically sourced, the analysis is unbiased, and all procedures align with established ethical guidelines.

Acknowledgments

This work was partially supported by Hong Kong RGC GRF No. 14206324, CUHK direct grant No. 4055209, and CUHK Knowledge Transfer Project Fund No. KPF23GWP20.

References

- AI@Meta. 2024. [Llama 3 model card](#).
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Sheng-guang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.
- Simon Baron-Cohen. 1991. Precursors to a theory of mind: Understanding attention in others. *Natural theories of mind: Evolution, development and simulation of everyday mindreading*, 1:233–251.
- Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. [MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.
- Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. 2018. Wizard of wikipedia: Knowledge-powered conversational agents. *arXiv preprint arXiv:1811.01241*.

- Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.
- Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. 2023. [Llm as os, agents as apps: Envisioning aios, agents and the aios-agent ecosystem](#). *Preprint*, arXiv:2312.03815.
- Gerd Gigerenzer. 1991. From tools to theories: A heuristic of discovery in cognitive psychology. *Psychological review*, 98(2):254.
- Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of The 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.
- Jie He, Jennifer Neville, Mengting Wan, Longqi Yang, Hui Liu, Xiaofeng Xu, Xia Song, Jeff Z. Pan, and Pei Zhou. 2025. GenTool: Enhancing Tool Generalization in Language Models through Zero-to-One and Weak-to-Strong Simulation. In *In the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*.
- Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. [Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios](#). *Preprint*, arXiv:2401.17167.
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. [Language models as zero-shot planners: Extracting actionable knowledge for embodied agents](#). *Preprint*, arXiv:2201.07207.
- Vojtěch Hudeček and Ondrej Dusek. 2023. [Are large language models all you need for task-oriented dialogue?](#) In *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 216–228, Prague, Czechia. Association for Computational Linguistics.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. 2023. [Mistral 7b](#). *Preprint*, arXiv:2310.06825.
- Mihir Kale and Abhinav Rastogi. 2020. [Template guided text generation for task-oriented dialogue](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6505–6520, Online. Association for Computational Linguistics.
- Wai-Chung Kwan, Huimin Wang, Hongru Wang, Zezhong Wang, Bin Liang, Xian Wu, Yefeng Zheng, and Kam-Fai Wong. 2024. [JoTR: A joint transformer and reinforcement learning framework for dialogue policy learning](#). In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 9578–9588, Torino, Italy. ELRA and ICCL.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich K  ttler, Mike Lewis, Wen tau Yih, Tim Rock  tschel, Sebastian Riedel, and Douwe Kiela. 2021. [Retrieval-augmented generation for knowledge-intensive nlp tasks](#). *Preprint*, arXiv:2005.11401.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. [API-bank: A comprehensive benchmark for tool-augmented LLMs](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.
- Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. [Code as policies: Language model programs for embodied control](#). *Preprint*, arXiv:2209.07753.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanwu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. [Agentbench: Evaluating llms as agents](#). *Preprint*, arXiv:2308.03688.
- Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2024. [Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities](#). *Preprint*, arXiv:2408.04682.
- Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. [Chameleon: Plug-and-play compositional reasoning with large language models](#). *Preprint*, arXiv:2304.09842.
- Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. [Agentboard: An analytical evaluation board of multi-turn llm agents](#). *Preprint*, arXiv:2401.13178.
- Gr  goire Mialon, Cl  mentine Fourier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. [Gaia: a benchmark for general ai assistants](#). *Preprint*, arXiv:2311.12983.

- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. [Webgpt: Browser-assisted question-answering with human feedback](#). *Preprint*, arXiv:2112.09332.
- Jeff Z. Pan, Siyana Pavlova, Chenxi Li, Ningxi Li, Yangmei Li, and Jinshuo Liu. 2018. Content based Fake News Detection Using Knowledge Graphs. In *Proc. of the International Semantic Web Conference (ISWC2018)*, pages 669–683.
- Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeliyanenko, Wen Zhang, Matteo Lissandrini, Russa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and Damien Graux. 2023. Large Language Models and Knowledge Graphs: Opportunities and Challenges. pages 1–38.
- J.Z. Pan, D. Calvanese, T. Eiter, I. Horrocks, M. Kifer, F. Lin, and Y. Zhao, editors. 2017a. *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Querying Answering*. Springer.
- J.Z. Pan, G. Vetere, J.M. Gomez-Perez, and H. Wu, editors. 2017b. *Exploiting Linked Data and Knowledge Graphs for Large Organisations*. Springer.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. [Gorilla: Large language model connected with massive apis](#). *Preprint*, arXiv:2305.15334.
- Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujuan Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. 2020. Few-shot natural language generation for task-oriented dialog. *arXiv preprint arXiv:2002.12328*.
- Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiushi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. [Toolrl: Reward is all tool learning needs](#). *Preprint*, arXiv:2504.13958.
- Libo Qin, Wenbo Pan, Qiguang Chen, Lizi Liao, Zhou Yu, Yue Zhang, Wanxiang Che, and Min Li. 2023a. [End-to-end task-oriented dialogue: A survey of tasks, methods, and future directions](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5925–5941, Singapore. Association for Computational Linguistics.
- Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023b. [Tool learning with foundation models](#). *Preprint*, arXiv:2304.08354.
- 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. 2023c. [Toolllm: Facilitating large language models to master 16000+ real-world apis](#). *Preprint*, arXiv:2307.16789.
- Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8689–8696.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. [Hugging-gpt: Solving ai tasks with chatgpt and its friends in hugging face](#). *Preprint*, arXiv:2303.17580.
- Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, Zhongyang Li, YE QI, Yang Ren, Dandan Tu, and Jeff Z. Pan. 2025. GeAR: Graph-enhanced Agent for Retrieval-augmented Generation. In *In the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*.
- Raphael Shu, Elman Mansimov, Tamer Alkhoul, Nikolaos Pappas, Salvatore Romeo, Arshit Gupta, Saab Mansour, Yi Zhang, and Dan Roth. 2022. [Dialog2api: Task-oriented dialogue with api description and example programs](#). *Preprint*, arXiv:2212.09946.
- Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.
- Ryuichi Takanobu, Hanlin Zhu, and Minlie Huang. 2019. [Guided dialog policy learning: Reward estimation for multi-domain task-oriented dialog](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 100–110, Hong Kong, China. Association for Computational Linguistics.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu,

- Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). *Preprint*, arXiv:2307.09288.
- Quan Tu, Shilong Fan, Zihang Tian, and Rui Yan. 2024. [Charactereval: A chinese benchmark for role-playing conversational agent evaluation](#). *Preprint*, arXiv:2401.01275.
- Dingmin Wang, Chenghua Lin, Qi Liu, and Kam-Fai Wong. 2021. [Fast and scalable dialogue state tracking with explicit modular decomposition](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 289–295, Online. Association for Computational Linguistics.
- Hongru Wang, Minda Hu, Yang Deng, Rui Wang, Fei Mi, Weichao Wang, Yasheng Wang, Wai-Chung Kwan, Irwin King, and Kam-Fai Wong. 2023a. [Large language models as source planner for personalized knowledge-grounded dialogues](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9556–9569, Singapore. Association for Computational Linguistics.
- Hongru Wang, Cheng Qian, Wanjuan Zhong, Xiushi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025a. [Otc: Optimal tool calls via reinforcement learning](#). *Preprint*, arXiv:2504.14870.
- Hongru Wang, Yujia Qin, Yankai Lin, Jeff Z. Pan, and Kam-Fai Wong. 2024a. [Empowering large language models: Tool learning for real-world interaction](#). In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24*, page 2983–2986, New York, NY, USA. Association for Computing Machinery.
- Hongru Wang, Huimin Wang, Zezhong Wang, and Kam-Fai Wong. 2022a. Integrating pretrained language model for dialogue policy evaluation. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6692–6696. IEEE.
- Hongru Wang, Lingzhi Wang, Yiming Du, Liang Chen, Jingyan Zhou, Yufei Wang, and Kam-Fai Wong. 2023b. [A survey of the evolution of language model-based dialogue systems](#). *Preprint*, arXiv:2311.16789.
- Hongru Wang, Rui Wang, Fei Mi, Yang Deng, Zezhong Wang, Bin Liang, Ruifeng Xu, and Kam-Fai Wong. 2023c. [Cue-CoT: Chain-of-thought prompting for responding to in-depth dialogue questions with LLMs](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12047–12064, Singapore. Association for Computational Linguistics.
- Hongru Wang, Rui Wang, Boyang Xue, Heming Xia, Jingtao Cao, Zeming Liu, Jeff Z. Pan, and Kam-Fai Wong. 2024b. [AppBench: Planning of multiple APIs from various APPs for complex user instruction](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15322–15336, Miami, Florida, USA. Association for Computational Linguistics.
- Hongru Wang, Boyang Xue, Baohang Zhou, Tianhua Zhang, Cunxiang Wang, Huimin Wang, Guanhua Chen, and Kam-Fai Wong. 2025b. [Self-DC: When to reason and when to act? self divide-and-conquer for compositional unknown questions](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6510–6525, Albuquerque, New Mexico. Association for Computational Linguistics.
- Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022b. [ScienceWorld: Is your agent smarter than a 5th grader?](#) In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024c. [Mint: Evaluating llms in multi-turn interaction with tools and language feedback](#). *Preprint*, arXiv:2309.10691.
- Zekun Moore Wang, Zhongyuan Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhao Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Man Zhang, Zhaoxiang Zhang, Wanli Ouyang, Ke Xu, Wenhu Chen, Jie Fu, and Junran Peng. 2023d. [Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models](#). *Preprint*, arXiv:2310.00746.
- Heng-Da Xu, Xian-Ling Mao, Puhai Yang, Fanshu Sun, and Heyan Huang. 2024. [Rethinking task-oriented dialogue systems: From complex modularity to zero-shot autonomous agent](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2748–2763, Bangkok, Thailand. Association for Computational Linguistics.
- Xinchao Xu, Zhibin Gou, Wenquan Wu, Zheng-Yu Niu, Hua Wu, Haifeng Wang, and Shihang Wang. 2022.

- Long time no see! open-domain conversation with long-term persona memory. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2639–2650, Dublin, Ireland. Association for Computational Linguistics.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *Preprint*, arXiv:2406.12045.
- Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. 2020. MultiWOZ 2.2 : A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117, Online. Association for Computational Linguistics.
- Xiaoying Zhang, Baolin Peng, Kun Li, Jingyan Zhou, and Helen Meng. 2023. SGP-TOD: Building task bots effortlessly via schema-guided LLM prompting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13348–13369, Singapore. Association for Computational Linguistics.
- Jinfeng Zhou, Zhuang Chen, Dazhen Wan, Bosi Wen, Yi Song, Jifan Yu, Yongkang Huang, Libiao Peng, Jiaming Yang, Xiyao Xiao, Sahand Sabour, Xiaohan Zhang, Wenjing Hou, Yijia Zhang, Yuxiao Dong, Jie Tang, and Minlie Huang. 2023. Characterglm: Customizing chinese conversational ai characters with large language models. *Preprint*, arXiv:2311.16832.
- Qi Zhu, Kaili Huang, Zheng Zhang, Xiaoyan Zhu, and Minlie Huang. 2020. CrossWOZ: A large-scale Chinese cross-domain task-oriented dialogue dataset. *Transactions of the Association for Computational Linguistics*, 8:281–295.
- Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Preprint*, arXiv:2306.13304.

Appendix

A Related Work

Dialogue System. On the one hand, task-oriented dialogue systems typically comprise four components Natural Language Understanding (NLU) (Goo et al., 2018), Dialogue State Tracking (DST) (Wang et al., 2021), Dialogue Policy Learning (DPL) (Wang et al., 2022a; Takanobu et al., 2019), and Natural Language Generation (NLG) (Peng et al., 2020), or adopt end-to-end framework (Qin et al., 2023a) to complete user goal through complex multi-turn interaction with various services in real-world. On the other hand, open-domain dialogue systems mostly follow retrieval-augmented generation framework (Lewis et al., 2021; Wang et al., 2023a) to retrieve different external knowledge such as persona (Xu et al., 2022) and document (Dinan et al., 2018), aiming to provide more personalized and informative responses.

Language Agent. To evaluate the effectiveness of LLMs as agents, many prior works have proposed various evaluation benchmarks (Liu et al., 2023; Ma et al., 2024; Sumers et al., 2023). For instance, VirtualHome (Puig et al., 2018) functions as a simulation platform for typical household activities, while ScienceWorld (Wang et al., 2022b) assesses agents’ scientific reasoning abilities within an interactive text environment. Distinguishing from these benchmarks, we target to build a virtual mobile environment where the LLMs need to call different APIs in various APPs in order to complete the complex goal of users via multi-turn dialogue interaction. Furthermore, our proposed benchmark can extend to multi-agent scenarios by conceptualizing large language models (LLMs) as operating systems (OS) and various applications (APPS) as distinct agents within the system (Ge et al., 2023). Besides that, assign agents some pre-defined roles has been proved an effective way to engaging user, resulting in more longer interaction time, such as character.ai, and also elicit some reasoning and role-specific capabilities of LLMs (Zhou et al., 2023). For example, RoleLLM (Wang et al., 2023d) and CharacterEval (Tu et al., 2024) evaluate the LLMs to generate role-consistent responses according to given background config of the role. On the contrast, we mainly focus on the language style of different roles (Zhou et al., 2023), leading to more personalized and acceptable response for users.

B Dataset Details

B.1 APP and API List

Table 6 show the full list of supported APP and API.

B.2 DialogTool Dataset

Table 7 show the full list of all actions in the DialogTool.

B.3 Schema of App and API

```
"Rents": {
  "desc": "a leading global provider of
    car rental solutions",
  "base_required_arguments": {},
  "APIs": {
    "getcarsavailable": {
      "desc": "discover cars available
        for rent in a certain location
        and period",
      "is_transactional": "False",
      "additional_required_arguments": {
        "city (str)": "city where you
          want to rent the car",
        "start_date (date)": "the first
          date to start using the
          rental car, the format
          follows yyyy-mm-dd.",
        "pickup_time (time)": "time for
          the pick-up, the format
          follows hh:mm",
        "end_date (date)": "the date to
          return the car, the format
          follows yyyy-mm-dd"
      },
      "optional_arguments": {
        "car_type (str)": "type of the
          car, value can only be one
          of follows: Hatchback, Sedan
          or SUV"
      },
      "result_arguments": {
        "car_type (str)": "type of the
          car, value can only be one
          of follows: Hatchback, Sedan
          or SUV",
        "car_name (str)": "car model",
        "pickup_location (str)": "place
          to pick up the car",
        "start_date (date)": "the first
          date to start using the
          rental car, the format
          follows yyyy-mm-dd",
        "pickup_time (time)": "time for
          the pick-up, the format
          follows hh:mm",
        "city (str)": "city where you
          want to rent the car",
        "end_date (date)": "the date to
          return the car, the format
          follows yyyy-mm-dd",
        "price_per_day (int)": "the cost
          for renting the car per day"
      }
    }
  },
}
```

APP	API
Banks	CheckBalance, TransferMoney
Buses	BuyBusTicket, FindBus
Events	FindEvents, GetEventDates, BuyEventTickets
Flights	SearchRoundtripFlights, ReserveRoundtripFlights, SearchOnewayFlight, ReserveOnewayFlight
Homes	FindHomeByArea, ScheduleVisit, FindApartment
Hotels	ReserveHotel, SearchHouse, BookHouse, SearchHotel
Media	FindMovies, PlayMovie, RentMovie
Movies	FindMovies, BuyMovieTickets, GetTimesForMovie
Music	PlayMedia, LookupSong, LookupMusic, PlaySong
RentalCars	GetCarsAvailable, ReserveCar
Restaurants	ReserveRestaurant, FindRestaurants
RideSharing	GetRide
Services	FindProvider, BookAppointment
Travel	FindAttractions
Weather	GetWeather
Calendar	GetEvents, AddEvent, GetAvailableTime
Alarm	GetAlarms, AddAlarm
Messaging	ShareLocation
Payment	MakePayment, RequestPayment
Trains	FindTrains, GetTrainTickets

Table 6: APP and API list.

Name	Desc	Type
Request	Request the value of an argument from the user	<i>tool-related</i>
Confirm	Confirm the value of all arguments before making a transactional API call	<i>tool-related</i>
Inform_Count	Inform the number of items found that satisfy user's request	<i>tool-related</i>
Notify_Success	Inform the user that their request was successful	<i>tool-related</i>
Notify_Failure	Inform the user that their request failed	<i>tool-related</i>
Inform	Inform the value for an argument to the user	<i>non-tool</i>
Offer_Intent	Offer a new intent to the user. Eg, "Would you like to reserve a table?"	<i>non-tool</i>
Req_more	Asking the user if they need anything else	<i>non-tool</i>
Goodbye	End the dialogue	<i>non-tool</i>

Table 7: Pre-defined Actions in SGD dataset. The name and descriptions are copied from SGD dataset (Rastogi et al., 2020).

```

"reservecar": {
  "desc": "make a rental car
    reservation",
  "is_transactional": "True",
  "additional_required_arguments": {
    "pickup_location (str)": "place
      to pick up the car",
    "start_date (date)": "the first
      date to start using the
      rental car, the format
      follows yyyy-mm-dd",
    "pickup_time (time)": "time for
      the pick-up, the format
      follows hh:mm",
    "end_date (date)": "the date to
      return the car, the format
      follows yyyy-mm-dd",
    "car_type (str)": "type of the
      car, value can only be one
      of follows: Hatchback, Sedan
      or SUV",
    "add_insurance (bool)": "whether
      to purchase insurance, True
      or False"
  },
  "optional_arguments": {},
  "result_arguments": {
    "car_type": "type of the car,
      value can only be one of
      follows: Hatchback, Sedan or
      SUV",
    "car_name": "car model",
    "pickup_location": "place to
      pick up the car",
    "start_date": "the first date to
      start using the rental car",
    "pickup_time": "time for the
      pick-up",
    "end_date": "the date to return
      the car",
    "price_per_day": "the cost for
      renting the car per day",
    "add_insurance": "whether to
      purchase insurance"
  }
}

```

```

},
"getride": {
  "desc": "book a cab for any
    destination, number of seats
    and ride type",
  "is_transactional": "True",
  "additional_required_arguments": {
    "destination (str)": "
      destination address or
      location for cab",
    "number_of_seats (int)": "number
      of seats to reserve in the
      cab",
    "ride_type (str)": "type of cab
      ride"
  },
  "optional_arguments": {},
  "result_arguments": {
    "destination": "destination
      address or location for cab
      ",
    "ride_type": "type of cab ride,
      value can only be one of
      follows: Pool, Regular or
      Luxury",
    "ride_fare": "total fare for cab
      ride",
    "wait_time": "expected waiting
      time for pick-up by cab",
    "number_of_seats": "number of
      seats to reserve in the cab"
  }
}
}
}

```

Figure 5: This is a sample JSON configuration of Rents App which contains 3 distinct APIs. We also provide name, format and possible values for categorical arguments. In this app, the base required arguments are empty.

C Prompt Details

You are a helpful assistant and you are good at Python.

Given the description, required arguments, optional required arguments and returned arguments of an APIs, generate a executable python code which implements the API.

Here is an example:

API:

```

getcarsavailable: {
  "desc": "discover cars
    available for rent in a
    certain location and period
    ",
  "is_transactional": False,
  "additional_required_arguments": {
    "city (str)": "city where
      you want to rent the
      car",

```

```

    "start_date (date)": "the
      first date to start
      using the rental car,
      the format follows yyyy
      -mm-dd.",
    "pickup_time (time)": "
      time for the pick-up,
      the format follows hh:
      mm",
    "end_date (date)": "the
      date to return the car,
      the format follows
      yyyy-mm-dd"
  },
  "optional_arguments": {
    "car_type (str)": "type of
      the car"
  },
  "result_arguments": {
    "car_type (str)": "type of
      the car",
    "car_name (str)": "car
      model",
    "pickup_location (str)":
      "place to pick up the
      car",
    "start_date (date)": "the
      first date to start
      using the rental car,
      the format follows yyyy
      -mm-dd",
    "pickup_time (time)": "
      time for the pick-up,
      the format follows hh:
      mm",
    "city (str)": "city where
      you want to rent the
      car",
    "end_date (date)": "the
      date to return the car,
      the format follows
      yyyy-mm-dd",
    "price_per_day (int)": "
      the cost for renting
      the car per day"
  }
}
}

```

Python:

```

def getcarsavailable(self, city,
  start_date, end_date,
  pickup_time, car_type=""):
  print("This is api [
    getcarsavailable] in [Rents
    ] app")
  results = []
  for db_sample in self.db["
    getcarsavailable"]:
    if db_sample["city"] in
      city and db_sample["
        start_date"] ==
        start_date and
        db_sample["end_date"]
        == end_date and \
        db_sample["pickup_time
          "] == pickup_time:
      if len(car_type) > 0
        and db_sample["

```

```

        car_type"] ==
        car_type:
            results.append(
                db_sample)
    elif len(car_type) ==
        0 or car_type is
        None:
            results.append(
                db_sample)

    return results

API: {api_desc}
Python:

```

Figure 6: The prompt used to prompt LLM to create tool in python code.

Given a dialogue between user and dialogue system, and a role config for dialogue system, please assign a consistency score according to all utterances by the dialogue system. The consistency of a role refers to the character's actions, dialogues, and decisions with their defined traits and background. The criteria for measuring "Consistency" are detailed in the following dimensions:

1. Behavioral Consistency: Evaluate whether the character's behavior aligns with their described personality and background across these aspects:
 - Personality Display: Does the character exhibit personality traits in interactions that match their predefined descriptions? For example, if a character is described as brave, they should exhibit bravery in the face of danger.
 - Background Response: Does the character's behavior in specific situations reflect their background knowledge and experiences? For instance, a character who was once stranded on a deserted island might display enhanced survival skills in similar settings.
 - Emotional Consistency: Do the character's emotional responses align with the situation and their personal history?
2. Dialogue Consistency: Assess if the character's dialogue reflects their personality traits and background story:
 - Language Style: Does the character's use of language suit their cultural and educa-

tional background?

- Relevance to Theme: Are the contents of the dialogue relevant to the character's life experiences and current situation?
- Emotional Expression: Does the character's emotional expression in dialogue match the personality described?

3. Decision-Making Consistency: Evaluate whether the character's decisions align with their goals and role setting:

- Goal Orientation: Do the decisions help the character achieve their set objectives?
- Background Logic: Do the decisions take into account the character's personal and societal background?
- Situational Appropriateness: Are the decisions reasonable and effective within specific scenarios?

Your output should range from 0 to 10, where 0 represents complete inconsistency and 10 represents perfect consistency. You only need to generate one score considering above factors without generating other information.

Figure 7: The prompt used to prompt LLM to assign the role consistency score.

Please determine which action of system should be invoked to generate the next response. Note some actions do not require the involvement of functional APIs. Here are all pre-defined actions starting from action_name followed by descriptions: {sys_actions}

You should output it in the format of [action_name(explanations)]. Your output should start with a square bracket '[' and end with a square bracket ']'. Do not output any other explanation or prompt. The action_name can only be one of pre-defined actions. You only need to choose one action according to what user needs.

Figure 8: The prompt used to prompt LLM to generate an action.

Given the API description and the existing dialogue history, please generate one API

request that should be invoked to complete the user's current query, and output it in the format of [api_name(#argument_1='value of argument_1', #argument_2='value of argument_2', ...)].

Here is the description of all APIs: {api_desc}

Here is the current date: {date}, make sure the values of all date related arguments is based on current date.

The api_name can only be one of pre-defined apis. You need to list all additional_required_arguments in the corresponding API, and optional_arguments when they are provided in the dialogue.

You should replace the value with the actual value in the dialogue context and attention on the format requirements of each argument. You can use "?" to replace the value when you can not infer it via current context.

Your output should start with a square bracket '[' and end with a square bracket ']'. Do not output any other unrelated explanation or tokens outside of [].

Figure 9: The prompt used to prompt LLM to generate API and all related arguments appeared in the dialogue in the required format.

Your task is to determine the required App according the description of each App and the last user turn in the dialogue.

Here is the information about all accessible Apps: {api_desc} Your output should follow the format [app1, app2, ...]. You only need to output one App.

Figure 10: The prompt used to prompt LLM to decide App first under the *hierarchical* setting.

Given the API description and the existing dialogue history, please generate one API request that should be invoked to complete the user's current query, and output it in the format of [api_name(#argument_1='value of argument_1', #argument_2='value of argument_2', ...)]. The api_name can only be one of pre-defined apis. You need to list all

additional_required_arguments in the corresponding API, and optional_arguments when they are provided in the dialogue. You should replace the value with the actual value in the dialogue context and attention on the format requirements of each argument. You can use "?" to replace the value when you can not infer it via current context. Your output should start with a square bracket '[' and end with a square bracket ']'. Do not output any other unrelated explanation or tokens outside of [].

Here is the current date: {date}, make sure the values of all date related arguments is based on current date.

Here is the description of all APIs: {api_desc}

Figure 11: The prompt used to prompt LLM to decide API and corresponding arguments after decided App under the *hierarchical* setting.

Given the API description and the existing dialogue history, please generate one API request that should be invoked to complete the user's current query, and output it in the format of [app_name: api_name(#argument_1='value of argument_1', #argument_2='value of argument_2', ...)]. The app_name and api_name can only be one of pre-defined apps and one of pre-defined apis in the app. You need to list all additional_required_arguments in the corresponding API, and optional_arguments when they are provided in the dialogue. You should replace the value with the actual value in the dialogue context and attention on the format requirements of each argument. You can use "?" to replace the value when you can not infer it via current context. Your output should start with a square bracket '[' and end with a square bracket ']'. Do not output any other unrelated explanation or tokens outside of [].

Here is the current date: {date}, make sure the values of all date related arguments is based on current date.

Here is the description of all Apps: {app_api_desc}

Figure 12: The prompt used to prompt LLM to decide App, API and all related arguments at the same time under the *flat* setting.

D Analysis

D.1 Tool Selection

To explore the specific performance of different LLMs on each API, we provide the accuracy of each API for each LLM as shown in Figure 14. We could draw several conclusion from the results.

Generally, as the model size increases, it usually leads to better results, as validated in LLaMA2 and LLaMA3 series models. However, we find that QWen1.5-7B and QWen1.5-72B surprisingly achieves better performance considering their sizes and QWen1.5-14B is the worst. Specifically, **most of LLMs achieves higher performance at Homes, Hotels and Travel Apps, and lower performance at Events, Restaurants, and Rents Apps.** We attribute this to there are several confusing APIs in later Apps. For example, getcaravaivable and reservecar in the Rents App, the agent usually needs to call getcaravaivable first and then reservecar to fulfill the use task, however, these APIs share most of common arguments and agent may misunderstand the relationship across them. Furthermore, we also analyze the App of unnecessary API calls, and we find most of them comes from Buses, Rents and Trains, the most unnecessary API calls are getcarsavailable and buybusticket.

D.2 Tool Execution

Besides matching and missing analysis at the main experiments, we additionally provide specific analysis for keys and values in the arguments when there exists both predicted arguments and ground truth arguments under the same API. In other words, we do not consider the determined API to be wrong or empty and require the keys and values to be exact matches. Table 8 shows the final results. We firstly emphasize that it is relatively unfair to directly compare performance across different LLMs since the total number of samples is different due to the different setting. However, we can observe several trends: **Keys.** It is obvious that it is challenging for existing LLMs to recognize all arguments from the multi-turn dialogue even given all descriptions about the APIs. Almost all LLMs tend to miss some arguments except LLaMA2-70B, resulting in

higher missing errors. Upon further investigation, we've identified a pattern in the missing cases: they tend to occur when there are optional arguments that the agent fails to predict. Additionally, this issue arises when the agent predicts extra arguments, as it may assign default values to certain optional arguments, such as the current date, or assume the number of passengers is always one.

Values. Most of value mismatch comes from the time and location related keys, such as *pickup_time* and *from*. The timing issue primarily stems from incorrect formatting and erroneous reasoning based on the given current date. On the other hand, location issues are often due to commonsense knowledge errors or hallucinations. For example, the dialogue agent might predict an incorrect location or use commonly known aliases for cities (e.g., "NYC" for New York City, "LAX" for Los Angeles). These observations requires more attention to further improve the performance of dialogue agent.

D.3 Human Evaluation

We hire three well-educated master students and randomly sample 50 response for each model. They were then asked to assign a score to each response, ranging from 1 (extremely poor, such as totally unrelated or API error) to 5 (extremely good, such as all details about the arguments are clearly stated). We provided specific examples for each score to illustrate the subtle differences in assignment. We calculate the average score of these annotators.

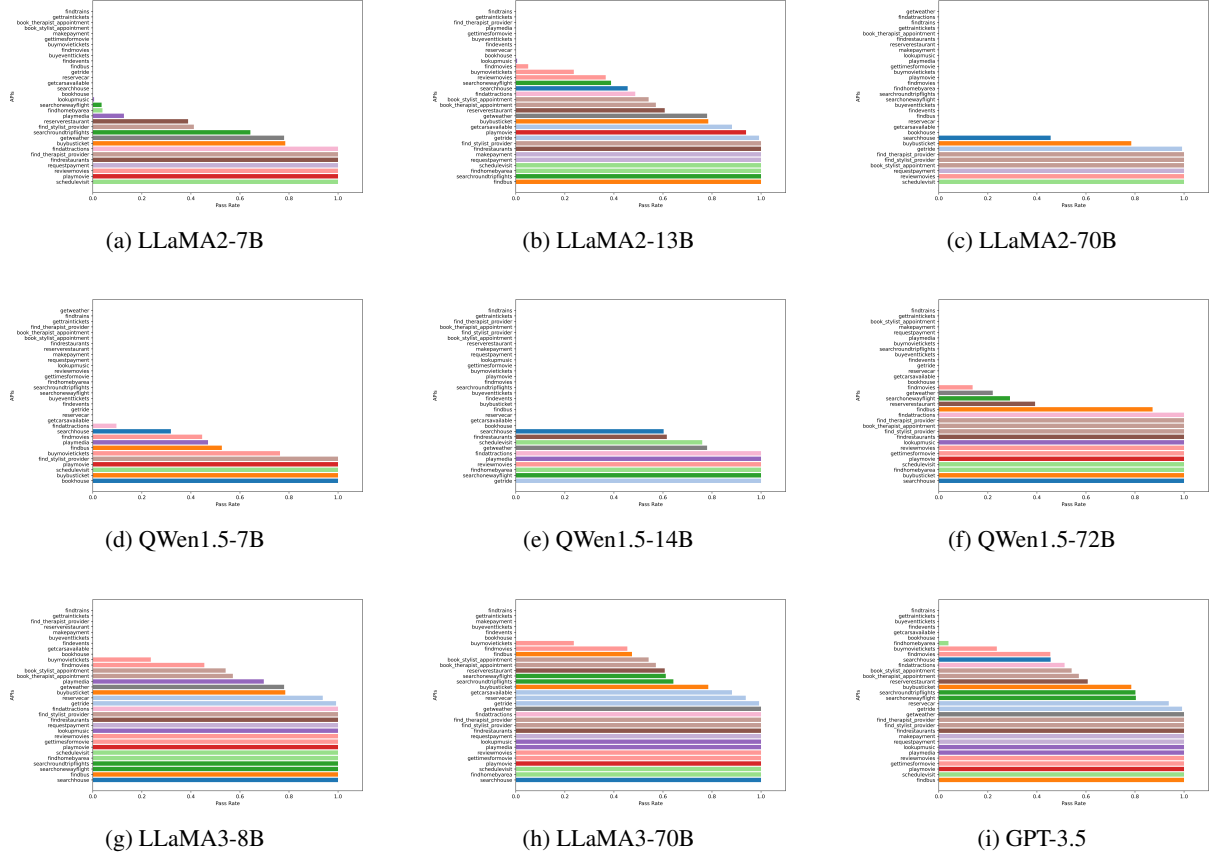
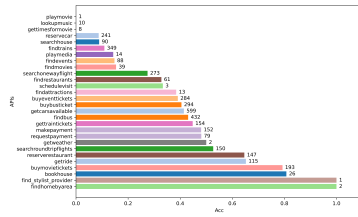


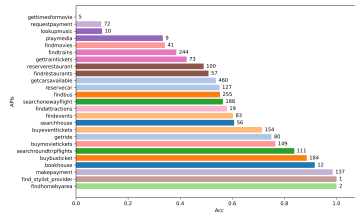
Figure 13: The tool creation performance of different LLMs on each API. We use same colour to indicate the API comes from same App.

Model	Miss (↓)	Extra (↓)	Value Mismatch (↓)
LLaMa3-8B	13.2	3.9	29.3 (65.7, 11.9)
LLaMA2-70B	19.0	27.6	41.6 (52.9, 15.4)
LLaMA3-70B	8.9	3.7	27.1 (70.6, 8.3)
GPT-3.5	18.5	4.0	23.5 (55.8, 20.6)
GPT-4o	12.4	0.9	24.3 (58.3, 16.3)

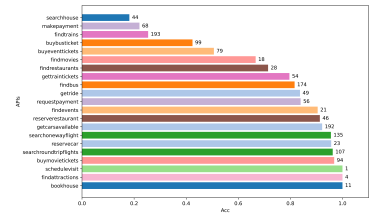
Table 8: Error Analysis of Tool Execution. It is worthy noting the Missing (Extra) column stands for the percentage of missing (extra) keys between predicted arguments and ground truth arguments. We also indicate two major types of value mismatch in (date-related mismatch, location-related mismatch).



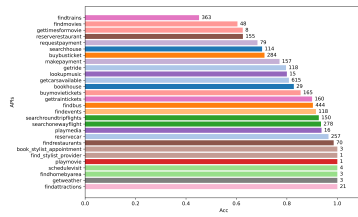
(a) LLaMA2-7B



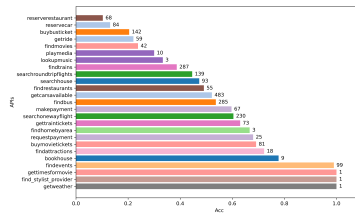
(b) LLaMA2-13B



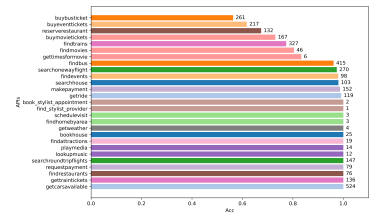
(c) LLaMA2-70B



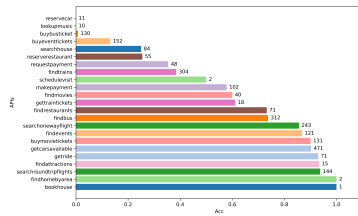
(d) QWen1.5-7B



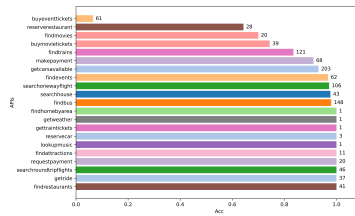
(e) QWen1.5-14B



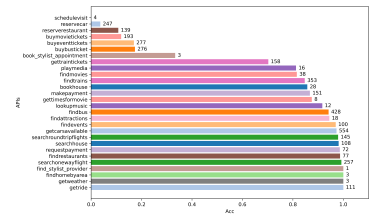
(f) QWen1.5-72B



(g) LLaMA3-8B



(h) LLaMA3-70B



(i) GPT-3.5

Figure 14: The performance of different LLMs on each API. We use same colour to indicate the API comes from same App. We also provide the frequency information at the end of bar.