# A Systematic Literature Review of Datasets and Benchmarks on Integrating LLMs with APIs

VITOR GABOARDI DOS SANTOS, Dublin City University (DCU), Ireland
BOUALEM BENATALLAH, Dublin City University (DCU), Ireland
FABIO CASATI, ServiceNow, Switzerland and University of Trento, Italy

Developing datasets and benchmarks for API-augmented Large Language Models (LLMs) is essential to support their integration with external tools and APIs. This systematic literature review analyzed 69 studies across three dimensions: integration tasks, dataset curation, and dataset characteristics. Results show that most datasets focus on API selection and API calling tasks, rely mostly on synthetic data, and often lack natural, diverse, and multi-turn interactions. The review recommends building richer and more realistic datasets that emphasize linguistic diversity, naturalness, context awareness, and hybrid curation combining human and synthetic data, together with improved evaluation methods for benchmarking API-augmented LLMs.

CCS Concepts: • **Computing methodologies** → **Natural language generation**; • **General and Reference** → **Surveys and overviews**; • **Software and its engineering** → *API languages.*

Additional Key Words and Phrases: LLM, API, Tool, Dataset, Benchmark, Review, Survey

## 1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable performance across many natural language processing (NLP) tasks, including text classification [86, 134], generation [6, 20], summarization [42], and translation [1]. One of their most widely used applications is conversational AI, where LLMs serve as chatbots and virtual assistants, enabling users to interact using natural language [17]. In this scenario, LLMs rely on their trained knowledge to generate responses. However, traditional LLMs face limitations in answering user utterances or requests that require real-time information (e.g., exchange rates, weather forecasts, flight prices) [59]; access to proprietary or personal data not included in the training data [32]; or complex mathematical reasoning.

Tool-augmented LLMs have emerged as a solution to overcome the above limitations by empowering LLMs with external tools [75]. Different approaches have been proposed to integrate external tools or data in LLM workflows. For instance, Retrieval Augmented Generation (RAG) uses retrieval mechanisms to obtain data from a knowledge base, database, or document corpus and incorporate it into the prompt, enabling the LLM to generate more context-aware responses [26]. OpenAI integrated ChatGPT with a browsing tool to obtain real-time information from the Web[1]. LLM-based agents include an action module that can interact with external tools to execute the agent's plan to fulfill the user utterance [105].

Furthermore, tool-augmented LLMs can interface with different types of external tools, including mainly: (1) APIs, (2) web and mobile applications, data sources and knowledge bases, and (3) external models [105]. APIs allow LLMs to control physical and digital environments (e.g., sending a command to a robot) or access external data (e.g., querying a currency API for current exchange rates). Data sources and knowledge bases can be queried using structured query languages to retrieve domain-specific or proprietary information. Also, external models can serve as specialized tools to

*This publication has emanated from research conducted with the financial support of Taighde Éireann-Research Ireland under Grant No. 18/CRT/6223.

[1]https://openai.com/index/introducing-chatgpt-search/

Authors' Contact Information: Vitor Gaboardi dos Santos, vitor.gaboardidossantos2@mail.dcu.ie, Dublin City University (DCU), Dublin, Ireland; Boualem Benatallah, boualem.benatallah@dcu.ie, Dublin City University (DCU), Dublin, Ireland; Fabio Casati, ServiceNow, Switzerland and University of Trento, Italy.

support complex tasks, such as generating and executing Python code [121, 137] or performing video summarization, image generation, or audio processing using domain-specific models [120].

APIs have been widely used as external tools in tool-augmented LLMs, primarily due to the plethora of available APIs [105]. For example, Toolformer [87] generated a dataset to train a model to decide when and how to call available APIs. Gorilla [67] fine-tuned a LLaMA-based model to generate API calls for utterances related to machine learning tasks. TaskMatrix.AI [54] presented an ecosystem that connects foundation models to millions of APIs to fulfill user utterances involving different inputs. Furthermore, as noted by Qu et al. [75], the definitions of "tool" and "API" often overlap. Some works treat each API as a standalone tool, while others define a tool as a collection of APIs. In this work, we adopt the former perspective, treating each API as an independent tool, which may consist of one or more API methods. While the term tool can encompass broader functionalities beyond APIs, our primary focus is on APIs. Therefore, we use the terms *API* and *tool* interchangeably throughout this work, unless otherwise specified.

The development of datasets that connect user utterances with APIs (hereafter referred to as API-augmented datasets) is essential for prompting, fine-tuning, or training LLMs to learn how to integrate utterances with the appropriate API. In parallel, benchmarks are needed to evaluate LLMs' ability to accurately interpret user utterances and interact with APIs reliably. As the demand for LLM-based agents grows, the development of high-quality datasets and benchmarks with tools that can obtain information and make decisions via APIs becomes critical to secure progress in this area. To address this need, this systematic literature review (SLR) investigates existing datasets and benchmarks designed to connect user utterances with APIs. By synthesizing the current landscape, we provide a resource for researchers and developers working on LLM-based agents, conversational interfaces, and other AI systems that rely on API integration. In summary, we aim to investigate the following three research questions (RQs):

- **RQ1. What are the tasks the datasets are designed to address?**
- **RQ2. How are the datasets curated?**
- **RQ3. What are the characteristics of the datasets?**

We searched for papers in key peer-reviewed conferences, journals, and ArXiv. We performed a thematic analysis [16] and identified several findings that we structured in a novel taxonomy: (1) datasets and benchmarks address different tasks when integrating LLMs with APIs, including API awareness, planning, API selection, API calling, intent recognition and response generation; (2) datasets and benchmarks are curated using different patterns, curators, quality control strategies, and API documentation leveraged from multiple sources; and (3) datasets and benchmarks have different characteristics in terms of scale, domain, API composition and user-LLM interaction mode. Lastly, while we acknowledge the existence of other surveys on the topic of tool learning with LLMs [72, 75, 93], our review focuses specifically on datasets and benchmarks, including an analysis of tasks, curation processes, quality attributes, and dataset structure.

The remainder of this SLR is organized as follows: Section 2 details the method used for selecting and analyzing relevant papers. Section 3 introduces the taxonomy we propose for categorizing research in this area. Sections 4, 5, and 6 present our findings on LLM-API integration tasks, dataset curation processes, and dataset characteristics, respectively. Section 7 presents a discussion summarising the main findings of the paper, key challenges in existing datasets, future research directions, and potential risks in the SLR. Finally, Section 8 presents the conclusion.

## 2 RESEARCH METHODOLOGY

Figure 1 summarises the method for selecting papers. This process consists of three main steps: (1) the search strategy, covering the construction of the search query and paper collection process; (2)

the definition of inclusion and exclusion criteria; and (3) the screening process to select the papers. We describe each step in the remainder of this section.

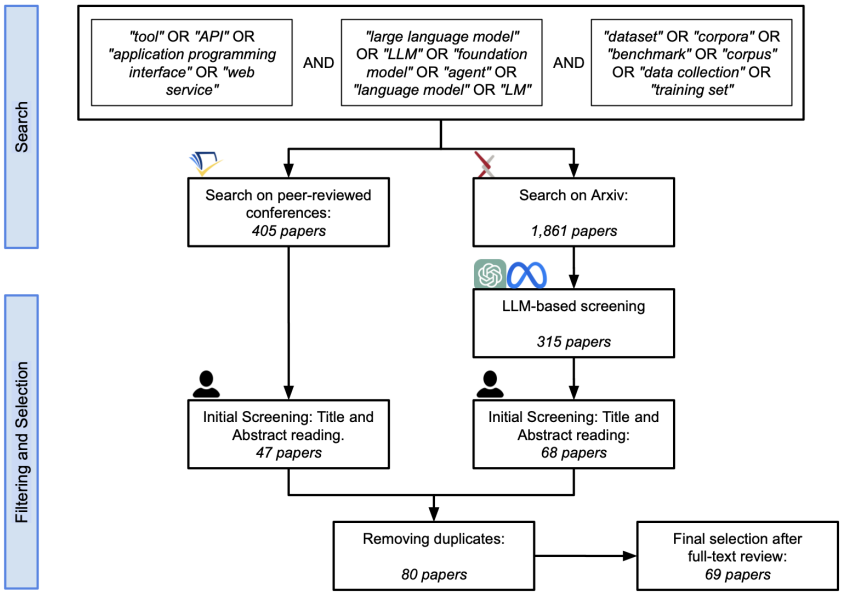



Fig. 1. Paper selection process.

## 2.1 Search Query

We used the following search query: ("tool" OR "API" OR "application programming interface" OR "web service") AND ("large language model" OR "LLM" OR "foundation model" OR "agent" OR "language model" OR "LM") AND ("dataset" OR "corpora" OR "benchmark" OR "corpus" OR "data collection" OR "training set"). This search query was structured around three main terms related to our topic — tool, LLM, and dataset — along with their synonyms to account for variations in terminology used in the literature.

To identify relevant papers, we searched three sources: peer-reviewed conferences and journals, and ArXiV. We selected a set of peer-reviewed conferences that, to the best of our knowledge, publish research on LLMs: NeurIPS, ICLR, EMNLP, NAACL, ACL, AAAI, COLING, IJCAI, ICSE, ASE, and FSE. The Semantic Scholar API [2] was used to collect papers from these venues. In addition, given the rapidly evolving nature of the field, many researchers publish their findings on ArXiV before submitting them to peer-reviewed venues. Therefore, we also searched for papers on ArXiV using their available API[3].

We applied date restrictions for two reasons: to maintain a manageable scope of the SLR and because the topic is relatively new, with generative LLMs gaining broader adoption after ChatGPT's launch in December 2022. Therefore, we considered peer-reviewed conferences from 2023 to 2024 and identified 405 papers. For ArXiV, we searched from January 2022 to September 2024 and found 1,861 papers.

[2]https://www.semanticscholar.org/product/api
[3]https://info.arxiv.org/help/api/index.html

### 2.2 Inclusion and Exclusion Criteria

We focused on datasets that connect LLMs with APIs in English. Papers introducing techniques without building a dataset or benchmark were excluded. Moreover, we excluded studies focused on multimodal models, code generation, text-to-SQL generation, mathematical applications, and retrieval augmented generation (RAG). These exclusions were made because most of these works do not link user utterances with APIs but instead focus on code development or reasoning without calling APIs, which fall outside the scope of this review.

### 2.3 Screening Process

We combined efforts from LLMs and humans to identify relevant studies. First, we discuss the use of GPT-4o and LLaMA-3.1-70B to assist in the initial screening of ArXiV papers. Next, we describe the manual screening process of both ArXiV and peer-reviewed papers. Finally, we explain the final selection of the paper.

**LLM-Based Screening**. Most of the papers retrieved from ArXiV were unrelated to the topic of the SLR, largely due to the broad use of terms like "*tool*" and "*dataset*" in LLM research. While these keywords are relevant to our topic, they are applied across many domains, leading to many false positives, an issue commonly encountered in systematic literature reviews [81]. Recent studies leveraged LLMs to assist in the screening process to mitigate the time associated with manual screening in SLRs, [43, 88]. For instance, Joos et al. [43] demonstrated that LLMs such as GPT-4o, LLaMA-3, and Claude-3.5-Sonnet can significantly accelerate the literature review process while maintaining recall at or below the typical human error threshold when selecting papers in the computer science domain. Thus, we employed GPT-4o and LLaMA-3.1-70B to assist in the screening process. For each paper found in ArXiV, we provided the title and abstract and asked the models to determine whether the paper "*introduce or propose a dataset or benchmark that can be used to connect LLMs with external tools or APIs*". To offer additional context, we included a one-shot example of a relevant paper to the SLR. We also deliberately omitted the exclusion criteria to allow the models to return a broad set of potentially related work. After running these prompts, GPT identified 145 relevant papers, while LLaMA identified 311, resulting in a total of 315 papers that at least one of the models considered relevant.

**Manual Screening**. A manual screening was conducted for all papers found in peer-reviewed conferences (405) and the papers from ArXiV that were considered relevant by either GPT or LLaMA (315). Two annotators reviewed the title and abstract of each potentially relevant paper, labeling them as either relevant or not based on the predefined inclusion and exclusion criteria. Papers were included when both annotators agreed on their relevance. Disagreements were resolved through discussion between the annotators. This process resulted in 47 relevant papers from peer-reviewed conferences and 68 relevant papers from ArXiv.

**Removing duplicates and final selection**. A total of 35 papers from peer-reviewed conferences were also published in ArXiv. After removing these duplicates, 80 unique relevant papers remained. Each paper was then fully reviewed, revealing that 11 were outside the scope of the SLR, resulting in a final selection of 69 papers.

## 3 TAXONOMY

In this section, we present the proposed taxonomy identified from this review, generated using thematic analysis [16]. Figure 2 shows a visual overview of the taxonomy. We structured the taxonomy according to the following dimensions that group datasets and benchmarks related to integrating LLMs with APIs:

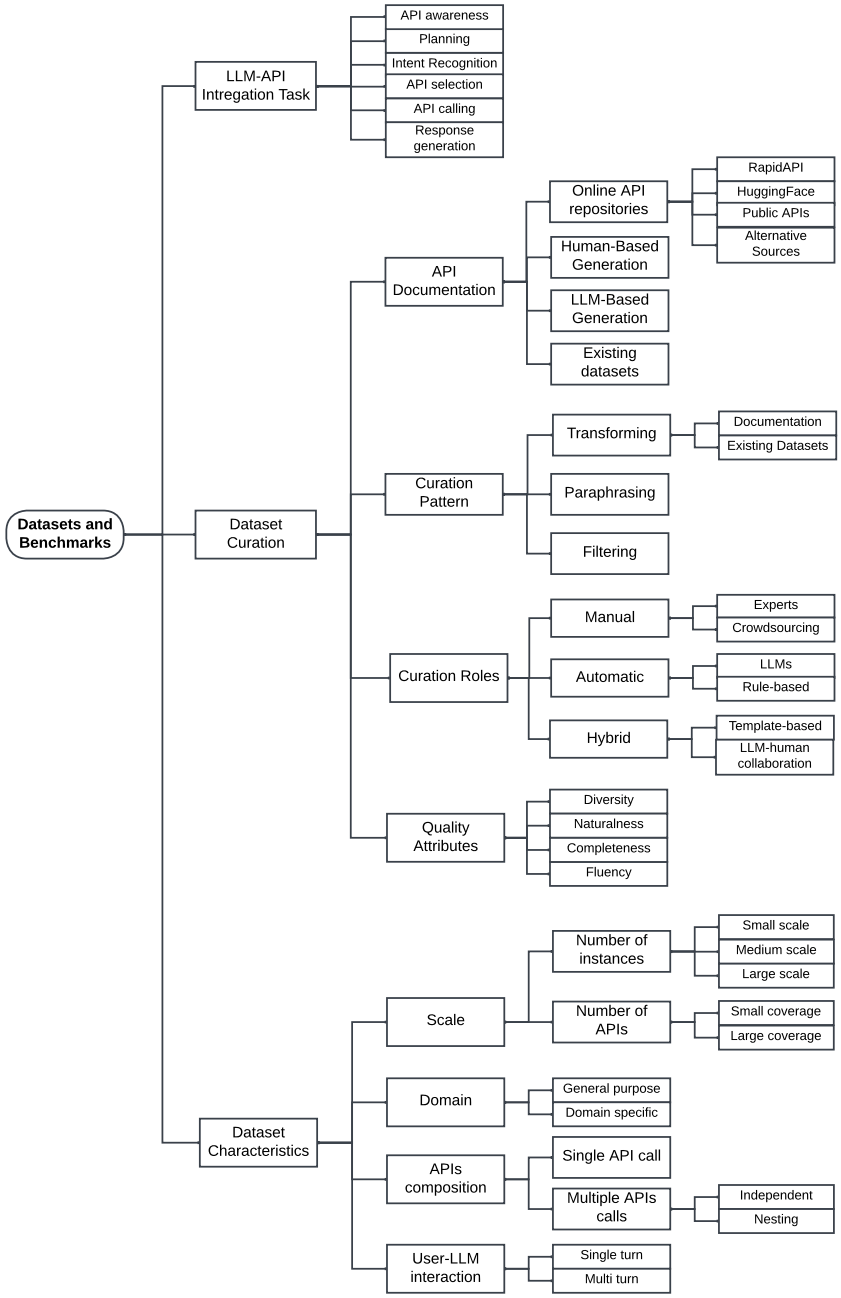



Fig. 2. Taxonomy of datasets and benchmarks for connecting LLMs with APIs.

- *LLM-API Integration Task* refers to the tasks a dataset is designed to address when evaluating, fine-tuning, or training models for LLM-API interaction. For instance, while some datasets

focus on planning a sequence of API calls to fulfill a user utterance, others focus on only selecting which API should be used.

- *Dataset Curation* refers to the process of collecting, generating, transforming, augmenting, or filtering data to build datasets that support training or fine-tuning LLMs to connect with APIs. We distinguish the following categories:
  - *API documentation* refers to the different sources used to obtain API documentation used to generate datasets. For example, API documentation can be collected from online repositories or synthetically generated using LLMs.
  - *Curation patterns* represent distinct strategies for creating API-related datasets. While some datasets are generated directly from API specifications, others involve paraphrasing existing API-related datasets or filtering instances from datasets based on specific criteria.
  - *Curation roles* refer to the individuals or systems responsible for the curation of the datasets. Datasets may be curated manually by experts or crowd workers, automatically using LLMs or rule-based programs, or a combination of both manual and automatic approaches.
  - *Quality attributes* describe different quality properties of the datasets. They are mostly used to guide the curation of datasets, affecting how data is created, filtered, or transformed. For example, some datasets prioritize linguistic diversity in user utterances, while others focus on including natural and realistic examples.
- *Dataset characteristics* describe the fundamental properties and structure of datasets. We identified the following categories:
  - *Scale* refers to the number of instances and APIs included in the dataset.
  - *Domain* represents the area that the dataset focuses on, which can be specific (e.g., focused only on travel, finance, or music), or general, covering multiple areas.
  - *User-LLM interaction mode* describes how messages are exchanged between users and LLMs. Datasets can have multiple-turn or single-turn conversations.
  - *APIs composition* refers to API flow that is involved in fulfilling the user utterance. Some datasets focus on a single API call for simple requests, while others involve invoking multiple API calls to handle more complex user utterances.

## 4 RQ1. LLM-API INTEGRATION TASKS

Our analysis reveals that datasets and benchmarks support six tasks: API awareness, planning, intent recognition, API selection, API calling, and response generation. Qu et al. [75] found that tool learning involves four tasks: tool planning, tool selection, tool calling, and response generation. We used this finding as a starting point and found two additional tasks: API awareness and intent recognition.

### 4.1 API awareness

API awareness refers to an LLM's ability to recognize whether a user utterance can be answered using its internal knowledge or requires external tools [27]. This awareness is crucial not only for preventing issues like hallucination, where the LLM may produce inaccurate information [39], but also for ensuring that the LLM recognizes when external tools are needed. By understanding the limitations in answering user utterances that require external APIs, an LLM can determine when to rely on these APIs for tasks that cannot be solved independently.

For example, the utterance "*What's the weather in Dublin tomorrow?*" cannot be solved using the LLM's internal knowledge alone, as weather information requires real-time data. In this case, the LLM should identify the need for a weather tool to provide an accurate response. In contrast, the utterance "*What is the capital of France?*" can be answered using the LLM's pre-existing knowledge.

Furthermore, datasets designed for this task usually include utterances paired with binary labels (e.g., "*yes*" or "*no*") indicating whether an external API is required to fulfill the utterance.

We found some datasets focused on API awareness [14, 38, 39, 107, 119, 133]. ToolE [39] developed a dataset with utterances that require the use of external tools, and utterances that the LLM can fulfill using its internal knowledge. Yang et al. [119] removed required information (e.g., missing destination when booking a flight or currency when converting money) from user utterances taken from API-Bank [52] and ToolBench [73] datasets to test whether the LLM is aware of missing information. NoisyToolBench [107] includes user utterances where the LLM must determine not only whether an API is needed, but also whether an API from an API pool is available to fulfill the utterance. ToolBH [133] developed a dataset with user utterances, a plan to solve the utterance, and a predefined set of tools, which may fulfill the utterance. ToolAlign [14] consists of question-answer pairs, including utterances that require API use, do not require API use, or contain harmful questions (e.g., requesting someone's confidential information or engaging in illegal activities).

### 4.2 Planning

Users often present complex utterances that require the use of multiple APIs [80]. Such utterances require planning to break down such complex tasks into subtasks, a process known as tool planning or task decomposition [91]. This process relies on the model's reasoning capabilities to analyze user utterance and API documentation and infer the appropriate actions that should be executed [70, 128]. Furthermore, task decomposition requires the LLM to iteratively adjust the plan to ensure it successfully addresses the user utterance [82, 124].

Different techniques have been proposed to enhance LLM planning and responses to user utterances. Chain of Thought (CoT) [108] enhances the reasoning abilities of LLMs by breaking down problems into intermediate steps. ReAct [124] prompts LLMs to solve complex decision-making tasks by alternating between reasoning traces and actions. This allows the model to create, maintain, and adjust plans and interact with external tools to incorporate additional information when fulfilling the user's utterance. Reflexion [96] integrates linguistic feedback from prior failures as additional context for the LLM, allowing the model to refine its planning in subsequent trials and improve task performance. RestGPT [98] employs an iterative, API-focused coarse-to-fine planning approach that decomposes user utterances into sub-tasks, selects appropriate APIs for each sub-task, executes the API calls, and extracts relevant information from the responses. Depth First Search-based Decision Tree (DFSDT) [73] builds a decision tree where nodes represent steps in a plan. DFSDT explores multiple reasoning paths, choosing either to proceed along a promising path or to abandon an existing node and expand a new node, increasing the likelihood of finding a valid plan. Reverse Chain [132] performs API planning using a backward reasoning process. It generates API calls in reverse order - starting with the final API required to fulfill the utterance and working backward to determine the preceding API calls. Ask-before-plan [131] proposed a three-agent framework for planning: the first interprets utterances and asks clarifying questions; the second interacts with external tools to obtain relevant information; and the third generates the final plan to fulfill the user's utterance.

Datasets designed for planning tasks typically consist of user utterances paired with a description of the steps required to accomplish the task. Figure 3 illustrates a user utterance that is decomposed into two steps, each further divided into three sequential substeps. In this example, the user asks the LLM to convert money from one currency to another. The task must be handled in two steps: (1) finding the exchange rate, and (2) calculating the exchange amount.

We found some papers focus on generating datasets for training or testing API planning in LLMs [12, 38, 51, 52, 92, 103, 111, 112, 131, 132]. These datasets differ in structure, yet they share a common objective of assessing the planning abilities of LLMs. Ultratool [38] constructed a dataset

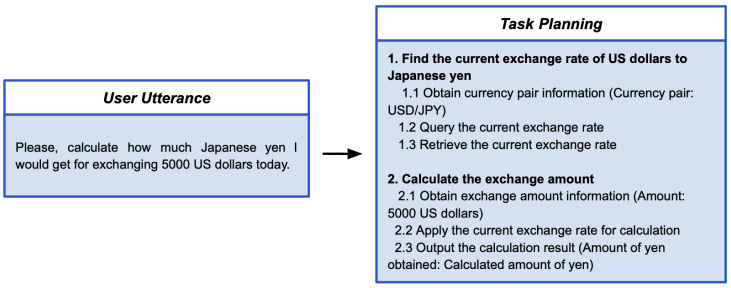



Fig. 3. Example of tool planning given a user utterance. Example adapted from UltraTool [38] dataset.

with pairs of user utterances and corresponding plan descriptions, where API method parameters are extracted directly from the user input. TaskBench [92] contains plans with textual descriptions of steps that include the names of the API methods that must be used in each step. T-Eval [12] contains utterances paired with sequences of API methods and their respective parameters. TARA [51] contains user utterances, responses, and a reasoning trace that outlines the thought process leading to the appropriate API selection. FlowBench [111] generated a benchmark to evaluate the planning capabilities of LLM-based agents under the assumption that the agent has prior knowledge of the required set of APIs. If a user utterance cannot be solved using the available set of APIs, the agent is expected to recognize this limitation and respond with an apology, stating that the utterance is beyond the scope of the agent.

### 4.3 Intent Recognition

Each user utterance or planned action can be classified into a relevant domain and a corresponding fine-grained intent before being mapped to a specific API [47]. The intent recognition task serves as a pre-processing step to streamline the API selection task (explained in the next section), narrowing down the API set required to fulfill the user utterance and improve system performance [24]. For instance, the utterance "*What's the weather in Dublin*" can be categorized under the "*weather*" domain with the intent class "*get_weather*". Given this classification, the tool selection module can select from multiple APIs within the weather domain (e.g., OpenWeatherMap API[4], Visual Crossing Weather API[5], or Weatherstack API[6]) to obtain the requested information.

We found that some studies incorporate intent recognition as part of the method for connecting LLMs with external APIs, as well as in the processes of dataset construction and benchmark evaluation [3, 11, 126, 128]. For instance, Re-Invoke [11] first extracts tool-related intents from the user utterance and then leverages the retrieved intents to identify relevant APIs that can fulfill the utterance. ALMITA [3] uses intents as seeds to generate an API-augmented dataset. MMAU [128] and ToolEyes [126] introduce benchmarks that assess essential capabilities of LLM-based agents, including their ability to understand user intent in tool-use applications.

Although intent recognition has traditionally been a fundamental component of task-oriented dialogue systems [47], most datasets designed for API-augmented LLMs do not explicitly address this task. Instead, they often treat each API as a distinct intent, directly retrieving APIs based on the user utterance and bypassing explicit intent classification.

[4]https://openweathermap.org/api
[5]https://www.visualcrossing.com
[6]https://weatherstack.com

### 4.4 API selection

API selection (also called API retrieval) is the process of identifying the top-most suitable APIs from a set of APIs that can be used to fulfill the user utterance [115, 136]. This task is particularly important when LLMs have access to a large number of APIs, making it impractical to include descriptions of all available APIs within a single prompt [75]. Failing to retrieve relevant APIs can lead to incomplete or inaccurate responses, leading to poor performance [52, 119].

We identified three primary approaches to API selection: LLM-based, embedding-based, and classification-based selection. In an LLM-based approach, the API documentation of multiple APIs is provided to the LLM along with the user utterance or planned step, and the LLM is prompted to select the most relevant APIs based on these descriptions. Several studies adopted this method [12, 21, 25, 38, 39, 58, 61, 83, 101, 132]. While leveraging the LLM's contextual understanding, this method is computationally expensive and constrained by token limits, which restrict the number of APIs included in prompts [11].

In embedding-based selection, an embedding model is used to compute vector representations of APIs, with each API represented by a vector stored in a database. During inference, the same embedding model encodes the user utterance and computes a similarity score (e.g., cosine similarity or dot product) between the utterance vector and all the API vectors stored in the database. The APIs with the highest similarity scores are selected as the most relevant [44].

The text used to compute API vectors changes across studies. In most cases, API documentation is used, which includes descriptions of API methods, parameters, and usage examples [29, 45, 60, 67, 73, 74, 113, 115]. However, semantic similarity in the vector space between user utterances and API descriptions may be low, which can impact retrieval effectiveness [64]. To mitigate this issue, some work generates synthetic utterances for APIs and creates an API vector using the generated utterances [19, 64]. Since a user's original utterance is semantically closer to similar natural language utterances than to formal API descriptions, this approach tends to improve retrieval performance. Re-Invoke [11] combines both approaches by constructing augmented API documentation, where synthetically generated utterances are concatenated with original API documentation. During API retrieval, the system compares the embedding of a user utterance against these augmented API vectors to identify the most relevant APIs. Retrievers are either fine-tuned on curated datasets [73, 74, 109] or rely on pre-trained embedding models [19, 45, 67].

In classification-based selection, an encoding-based model (e.g., DeBERTa or RoBERTa) is fine-tuned to categorize user utterances into a predefined set of API methods. We found a few works that adopted this approach [22, 64]. Although this approach can improve retrieval performance compared with others, it only supports a fixed set of API methods and requires additional fine-tuning when new API methods must be considered.

We found that 21 datasets reviewed in this study addresses the API selection task [5, 9, 13, 21, 22, 25, 33, 45, 58, 61, 64, 74, 90, 95, 98, 100, 101, 104, 114, 128, 132]. These datasets generally consist of user utterances paired with APIs or API methods. Some datasets, such as RestBench [98] and ToolFlow [95], pair utterances with multiple API methods of a single API. Other datasets incorporate additional elements besides pairs of utterances and APIs. Confucius [25] includes an utterance, response, and ground truth APIs, while ToolVerifier [61] introduces a candidate set of APIs along with a reasoning trace explaining the correct selection. ToolBench [73] introduces multi-intent utterances, which link utterances to different domains, APIs, and methods, thereby enabling more complex scenarios.

### 4.5 API calling

API calling (also known as slot or parameter filling) involves mapping parameters from the user utterance to the arguments of an API method, adhering to the rules provided in the API documentation [106]. This mapping poses challenges, as parameter format constraints may differ from how users express them in utterances [58]. For example, a user may specify a currency as "*US dollars*" while an API method requires the standardized format "*USD*". Once the API method's arguments are filled, the API call is invoked to obtain the necessary information [75]. Most API calls are generated using LLMs that take the user's utterance and the documentation of selected API methods as input.

We found that LLM-based parameter filling follows two main strategies: utterance-only extraction and context-aware extraction. In the utterance-only extraction strategy, the LLM retrieves all API method parameters only from the user utterance. The complexity of the utterance influences how many API calls are required: simple utterances may require a single API call [67, 73, 104, 128, 130], while more complex utterances require multiple API calls [33, 35, 73, 99, 100, 104]. We discuss APIs composition with more details in Section 6.4.

In the context-aware extration strategy, datasets provide additional contextual information (e.g., policies or user preferences and constraints) in addition to user utterances to generate API calls or fill in missing parameters [63, 66, 123, 127]. $\tau$-Bench [123] includes a policy document outlining domain-specific rules that LLMs must follow when interacting with users and filling API arguments. For instance, when booking flights, policies specify different baggage allowances based on membership status, requiring the LLM to use these rules to fill API method parameters aligned with the user's membership. Natural Language Standing Instructions (NLSI) [63] introduces standing instructions, which define user preferences and constraints when the user first use a natural language interface. The LLM fills API call arguments using these standing instructions when an utterance lacks essential information. For example, if a user includes a standing instruction stating the preference for Italian food, an utterance like "*Order something for me to eat*" should prompt the LLM to infer Italian cuisine instead of asking the user.

We found that many datasets focus on the API calling task [5, 9, 22, 33–35, 58, 66, 73, 90, 99, 100, 104, 113, 127, 128, 130, 132]. Most datasets represent API-calling instances using JSON files that include details such as the API name and URL, method name, and parameters extracted from the user utterance, mapped to API method parameters. Figure 4 illustrates an example adapted from the APIGen [56], where the user requests the weather forecast for New York City over the next five days, formatted in Celsius. To fulfill this utterance, the *OpenWeatherMap* API can be used with the *getforecastweather* method. The parameters, such as the location ("*q*": "*New York*"), temperature units ("*units*": "*metric*"), and forecast duration ("*cnt*": *5*), are mapped from the user utterance to the API method in JSON format. Other approaches represent API calls using Python function definitions [67, 132], cURL commands [31], or command line interfaces (CLI) [37].

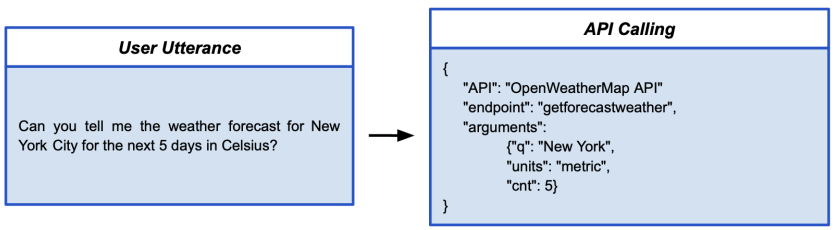



Fig. 4. Example of a tool calling given a user utterance. Example adapted from APIGen [56] dataset.

### 4.6 Response generation

API call responses are commonly formatted as JSON files containing the requested information, but their structure is difficult for end users to interpret. To enhance readability, the data is transformed into natural language text that directly answers the user's utterance.

There are multiple approaches to convert the JSON structured data into natural language text, including hand-engineered rules for generating fixed-format text, or probabilistic models for more flexible text generation [55]. Recently, LLMs have become widely used for this task [68]. In such cases, the API's JSON output is given to the LLM as context, and the LLM is prompted to generate a natural language response to the user's utterance. Furthermore, sometimes the LLM must interpret the API's output to provide an accurate response [94]. For instance, a weather API may provide forecasts for cities in California, but not directly filter for sunny weather. To answer an utterance like "*Which cities in California will have sunny weather tomorrow?*" the LLM would need to acquire forecasts for all cities and filter the results for sunny conditions.

We found that datasets focused on response generation tasks typically link a user utterance to a natural language text response that answers the utterance. Details such as the API description, API methods, parameters, and other metadata are often included. Figure 5 illustrates an example from AnyToolBench [21], including a user utterance, the relevant API, and the method required to answer the utterance, and the generated response created after processing the API call response.

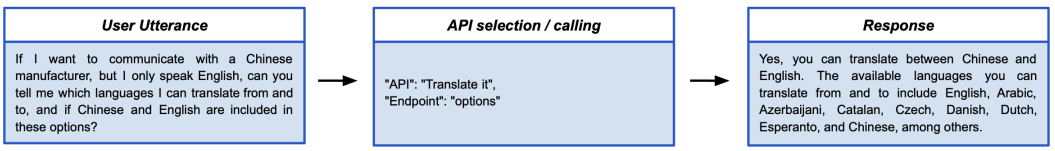



Fig. 5. Example of response generation given a user utterance. Example taken from AnyToolBench [21] dataset.

We also found a few datasets that focus on this task, since API responses can change depending on the time of the request. In general, if the API call is correct, it can be executed to retrieve information and answer the utterance. Nevertheless, some datasets provide the final response, allowing the evaluation of the entire pipeline without calling the API [21, 30, 51, 100, 107, 130]. For instance, AnyToolBench [21] leverages GPT-4 to first generate parameters and utterances for specific APIs. Then, they prompt GPT-4 again to generate a response for each utterance and verify whether the generated response solves the utterances. NoisyToolBench [107] includes responses that an LLM should ideally generate when handling user utterances with missing information, along with the corresponding answers to these clarifying questions from the user.

## 5 RQ2. DATASET CURATION

Our analysis shows four relevant aspects involved when curating datasets and benchmarks: the source of API documentation, curation patterns, curation roles, and quality attributes used to guide the data generation. We explain each aspect in the remainder of this section.

### 5.1 API documentation

API documentation is used as a reference for curating API-augmented datasets, commonly adhering to established standards, such as the OpenAPI Specification (OAS) [69]. However, other formats are also used, including JSON schemas, ChatGPT plugin descriptions, and Python function signatures. Based on our analysis, we found that API documentation is typically generated or obtained through

the following methods: from online API repositories, via human-written documentation, through LLM-based generation, or by leveraging existing datasets.

*5.1.1 Online API repositories.* We found that most approaches leverage API documentation from online API repositories [2, 4, 9, 31, 37, 39, 58, 64, 67, 73, 90, 92, 98, 109, 113, 114, 128]. We identified four main sources of online API repositories: RapidAPI[7], HuggingFace[8], Public APIs from official websites, and alternative sources.

RapidAPI serves as a platform for developers to discover, test, and connect to thousands of APIs. While it does not provide a standardized API documentation format, it offers a wealth of API-related data, making it an important resource for multiple works [4, 9, 58, 73, 128]. ToolBench [73] scraped information from RapidAPI, including API descriptions, method lists, and parameters, resulting in a dataset of over 16,000 APIs. NESTful [4] manually curated information for 39 APIs from RapidAPI, focusing on specifications such as API names, methods, and input parameters.

HuggingFace is a widely used platform in the machine learning and data science communities, enabling developers to host models, datasets, and model card information, which describes the functionality, usage, and input-output requirements of hosted models. Some work, such as Gorilla [67] and TaskBench [92], used model card information to create datasets that pair user utterances with API calls for machine learning-related tasks.

Other work gathers API information directly from public APIs provided on official websites [2, 19, 37, 64, 98, 113, 114]. ToolBench2 [114] collected API documentation from six different sources, including OpenWeather[9] and Google Sheets[10] APIs. RestBench [98] used documentation from Spotify and TMDB. Hsieh et al. [37] scraped API documentation about 200 APIs from the Google Cloud Platform[11].

Other works use alternative sources to obtain API documentation. ToolE [39] incorporated descriptions from OpenAI plugins. API Pack [31] gathered documentation from multiple platforms like RapidAPI, APIs.guru[12], Swagger Hub [13], and IBM API Hub[14]. ShortcutsBench [90] collected documentation by crawling websites that host user-created actions from Apple Shortcuts[15].

*5.1.2 Human-generated API documentation.* API documentation can be also manually created [12, 23, 25, 52, 66, 92, 99, 103, 123]. It ensures that datasets are tailored to particular domains, but it also introduces labor costs and requires specialized expertise to generate API documentation. For instance, T-Eval [12] included manually generated documentation of about 15 APIs. API-Bank [52] hired a team of senior researchers and development engineers to manually generate 73 API specifications for a testing subset of their dataset. TaskBench [92] manually created 40 API documentations focusing on domains such as education, travel planning, and online shopping. AppWorld [103] created detailed API documentation and the code that implements the APIs.

*5.1.3 LLM-generated API documentation.* Some datasets use LLMs to generate API documentation, reducing the effort required for manual creation [3, 33, 52, 61, 83, 100, 109]. We found that LLM-based API generation falls into two approaches: (1) generation based on existing API descriptions and (2) generation from predefined domains or structured taxonomies.

[7]https://rapidapi.com/
[8]https://huggingface.co
[9]https://openweathermap.org/api
[10]https://www.google.com/sheets/about/
[11]https://cloud.google.com/sdk/gcloud/reference
[12]https://apis.guru
[13]https://app.swaggerhub.com/search
[14]https://developer.ibm.com/apis/
[15]https://support.apple.com/en-ie/guide/shortcuts

LLMs are prompted to generate API documentation using as input API descriptions available from online repositories or manually generated examples to guide the generation [38, 52, 61, 100]. ToolAlpaca [100] and API-Bank [52] used API data from the Public APIs repository[16] to generate API documentation. ToolAlpaca [100] extracted API names and descriptions from the Public APIs repository and used LLMs to generate extended descriptions, function documentation, and parameter details. In contrast, API-Bank [52] used two LLM-based agents to generate domain-specific APIs using examples from the Public APIs repository. ToolVerifier [61] prompted LLaMA-65B to generate API documentation using manually generated API documentation as few-shot examples. In UltraTool [38], experts proposed an initial set of APIs and then prompted GPT-4 to expand them by adding additional methods.

Other works prompt LLMs to generate API documentation based on predefined domains or taxonomies [3, 33, 83, 109].Seal-tools [109] used ChatGPT to generate tools for domains defined by the authors, filling predefined templates with attributes such as API name, description, methods, and parameters. NesTools [33] generated API documentation formatted as Python functions using GPT-3.5 by feeding the same domains proposed by Seal-Tools [109]. ToolEmu [83] used GPT-4 to create API names, descriptions, and detailed specifications based on a taxonomy of 18 domains. Unlike SEAL-Tools, ToolEmu included manual refinement to ensure accuracy and relevance. ALMITA [3] used three sequential prompts in GPT to generate API documentation: first, to create customer support issues (e.g., order cancellation); second, to generate a procedure for resolving them; and third, to produce API documentation with methods that can be used to solve the procedure.

*5.1.4 Existing datasets.* We found that some datasets use API documentation from existing API-augmented datasets [13, 14, 21, 30, 35, 56, 62, 74, 107, 110, 115, 119, 130, 132, 135]. In summary, existing API documentation is refined, adapted, or expanded based on prior efforts. APIGen [56], NoisyToolBench [107], and ToolLens [74] used a subset of API documentation from the ToolBench dataset [73], which has been widely used across multiple works. APIGen [56] removed APIs with incorrect parsing, missing required or optional parameters, and API calls that failed execution, while NoisyToolBench [107] used a subset of 100 APIs. ToolPreference [8] leveraged both successful and failed API usage attempts from the decision tree paths in ToolBench [73] to build a novel API-augmented dataset. AgentInstruct [62] refined API documentation from different sources (e.g., code snippets and ToolBench [73]) by including additional method and parameter descriptions. Zhang et al. [132] used APIs from API-Bank [52] while OOD-Toolset [35] used APIs from ToolAlpaca [100].

## 5.2 Curation Patterns

Curation patterns refer to the different strategies used to build API-augmented datasets. In our analysis, we found three primary curation patterns: transforming, paraphrasing, and filtering. Figure 6 shows examples of each covering the generation of user utterances, which are explained in detail throughout this section.

*5.2.1 Transforming Pattern.* This pattern involves modifying or adapting existing API-related information (e.g., OAS, descriptive metadata, or existing datasets) to generate datasets or benchmarks. We categorize the transformation into two main sources: API documentation and cross-domain datasets.

**From API Documentation**. In this approach, API documentation is provided to a curator (e.g., LLM, experts, or crowd workers) to generate datasets. In most cases, OpenAPI Specifications (OAS) are presented, and curators write user utterances and API calls based on API descriptions. The API documentation discussed in Section 5.1 are used in this strategy. Figure 6 illustrates an example

[16]https://github.com/public-apis/public-apis

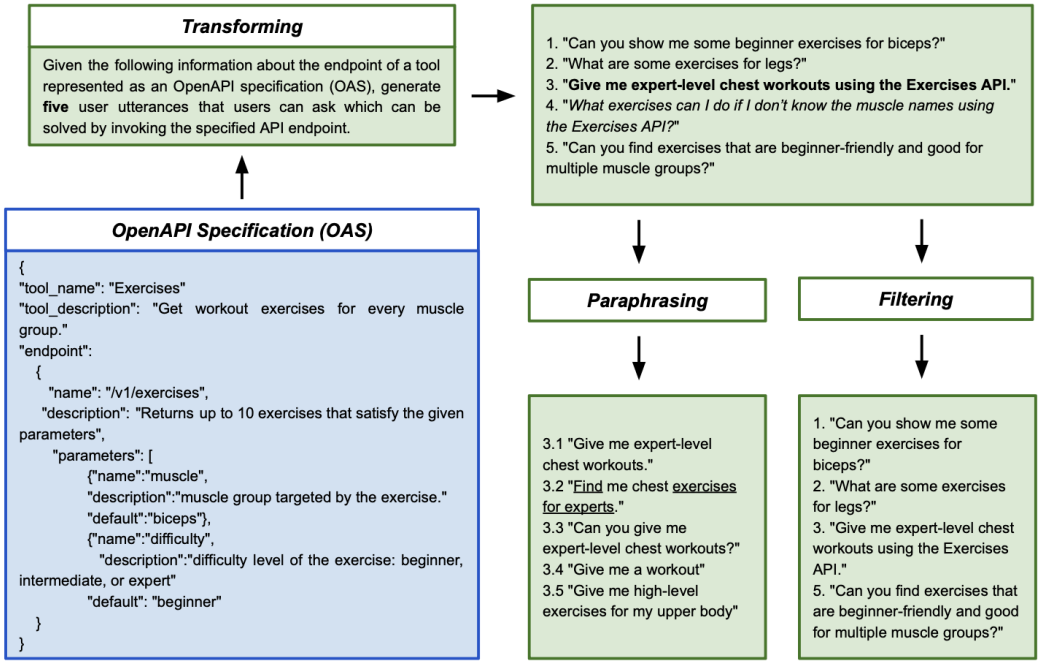



Fig. 6. Illustration of different curation patterns focused on creating user utterances.

where an OAS for retrieving workout exercises is given to a curator, who then generates five utterances that can be solved using the provided API. The prompt usually includes guidelines to improve the quality of the generated utterances.

We found that many work use API documentation to build API-augmented datasets [3, 22, 23, 25, 31, 33, 39, 52, 56, 58, 62, 66, 67, 73, 74, 90, 100, 128, 130, 132]. ToolBench [73] prompted GPT-3.5 to generate user utterances and API calls by feeding sampled API method descriptions and manually written examples. API-Pack [31] prompted the Mistral-7B model by feeding API method descriptions and few-shot examples to generate user utterances. ToolE [39] leveraged API names and descriptions from OpenAI's plugin list and used multiple prompts to generate utterances. ToolTalk [23] used API documentation to create descriptions of realistic user scenarios that represent users attempting to accomplish API-related tasks. These scenarios are used to guide the creation of multi-turn conversations between users and LLMs. Confucius [25] generated utterances by incorporating API documentation from multiple APIs and providing in-context demonstrations of user utterances and responses. ShortcutsBench [90] generated utterances by providing not only API descriptions but also API calls that should be used to fulfill the utterance.

**From cross-domain datasets**. In this approach, API-augmented datasets are created by adapting existing datasets from other domains (e.g., task-oriented chatbots, digital assistants). For instance, API-Blend [5] generated user utterances and API calls by repurposing datasets originally designed for tasks like semantic parsing, dialogue systems, and digital assistants [10, 71, 77, 125]. NLSI [63] and AppBench [104] generated API-augmented datasets from the Schema-Guided Dialogue (SGD) dataset [77], which includes conversations between a human and a virtual assistant. TARA [51] collected user utterances from datasets covering tasks such as translation [49] and general knowledge questions [46] to generate API calls.

*5.2.2 Paraphrasing Pattern.* This pattern rewrites or restructures utterances to capture the diverse ways users can write. The main goal is to evaluate or fine-tune API-augmented LLMs to handle variations in user expressions more effectively. Figure 6 illustrates an example of paraphrasing, where utterance number 3 (highlighted in bold) is rewritten in five different ways: removing the explicit mention of the Exercises API (utterance 3.1); replacing words such as "*give*" with "*find*", and "*expert-level workouts*" with "*exercises for experts*" (utterance 3.2); changing the syntax structure from imperative to a question (utterance 3.3); ommiting detail about the targeted muscle group (utterance 3.4); and adding ambiguity, as the term "*upper body*" may refer to the chest, shoulders, back, and arms, making it unclear which muscle group is the focus (utterance 3.5). Traditional paraphrasing techniques in NLP usually maintain the core intent of the seed utterance [117]. However, we adopt paraphrasing as a broader abstraction level, where modifications may intentionally be designed to change the original meaning. For instance, some datasets introduce ambiguity or remove essential information of the utterance to assess whether the LLM can still detect the missing information and request clarification [58, 62, 107, 131].

We found that many work uses the paraphrasing pattern [12–14, 19, 29, 38, 58, 58, 62, 74, 107, 110, 115, 119, 119, 131]. T-Eval [12] paraphrased utterances by incorporating guidelines to remove API names and avoid using fake user information (e.g., fake ID numbers like 123456). ToolE [39] replaced "ChatGPT" mentions in the utterance with "Chatbot". NoisyToolBench [107] modified utterances from ToolBench [73] by removing required parameters and adding ambiguous references that lead to multiple interpretations. Yang et al. [119] removed essential information required to fulfill user utterances from ToolBench [73] and API-Bank [52] datasets, aiming to assess whether LLMs can recognize incomplete conditions. ToolSandbox [58] paraphrased utterances that require multiple APIs or omit essential details. Santos et al. [19] generated paraphrases while encouraging different word choices. PPTC-R [135] modified utterances while preserving their original meaning by rewording with synonyms or altering syntax and adding irrelevant chitchat sentences. Agent-FLAN [13] converted utterances structured in "Thought-Action-ActionInput" templates into multi-turn conversations to generate more natural utterances that better reflect how users engage with conversational agents, preventing fine-tuned LLMs from overfitting to rigid formats (e.g., JSON or ReAct templates) presented in other datasets.

*5.2.3 Filtering Pattern.* This pattern removes utterances or API calls that do not meet predefined quality attributes (e.g., removing API calls that return errors or user utterances that are semantically unrelated to the API). The goal is to retain only high-quality instances, ultimately enhancing LLM performance when fine-tuned or trained on the dataset. Figure 6 illustrates an example of filtering, where utterance number four (highlighted in italics) is removed because it is confusing, lacks relevant information, and mentions an API name. The quality attributes used to filter instances change across works (discussed further in Section 5.4). While the *paraphrasing* pattern modifies utterances to meet quality criteria, the *filtering* pattern removes utterances that fail to meet attributes.

Multiple works used the filtering pattern to improve the quality of API-augmented datasets [9, 19, 30, 33, 35, 39, 41, 45, 51, 56, 104, 113, 127, 133]. Iskander et al. [41] filtered utterances from ToolBench [73] and ToolAlpaca [100] that failed to meet the following quality criteria: (1) utterances missing required parameters, (2) logically unrelated requests, and (3) APIs that could not fulfill the given utterance. The filtered dataset was used to fine-tune LLMs, resulting in better performance compared to fine-tuning on the full original dataset. Agent-FLAN [13] discarded utterances from the AgentInstruct [62] and ToolBench [73] that do not adhere to the "Thought-Action-ActionInput" format. AppBench [104] first assigned fluency and diversity scores to utterances on a scale of 1 to 10 and then filtered utterances with scores below 6. APIGen [56] removed API calls that failed JSON format checks or returned errors when executed, ensuring only valid and executable calls

remained. OOD-Toolset [35] discarded ambiguous utterances and API calls that resulted in errors when invoked. ToolE [39], ToolBH [133], and RoTBench [127] filtered duplicate utterances (i.e., highly semantically similar utterances), keeping only one representative example. In contrast, Santos et al. [19] removed paraphrases that were semantically unrelated to the seed utterance.

### 5.3 Curation Roles

The curation roles refer to the individuals or systems responsible for curating the datasets and benchmarks. We identified three curation roles, which change based on the level of human involvement in the curation process: manual, automatic, and hybrid.

*5.3.1 Manual.* This role relies on human expertise and creativity to generate datasets and benchmarks. It can be categorized into two strategies: expert-driven and crowdsourcing.

**Expert-driven** leverages the expertise of professionals such as developers, researchers, or computer science students, who use their knowledge and familiarity with APIs to curate datasets. In general, higher quality outputs are achieved due to the expertise of the curators. However, this approach is time-consuming and costly, making it difficult to scale for larger datasets.

We found some work curated datasets using human experts [35, 52, 58, 79, 85, 98, 107, 112, 126, 128]. RestBench [98] requested six experts on NLP to generate utterances and API calls. API-Bank [52] created a testing dataset by requesting computer science students to generate multi-turn dialogues and API calls based on randomly selected APIs. Two annotators evaluate the dataset to ensure quality. ToolSandbox [58] manually generated instances using two experts in tool-use scenarios. TravelPlanner [112] requested 20 graduate students to generate plans for solving user utterances. The authors then manually reviewed and refined each utterance and its corresponding plan. ToolFlow [95] requested seven NLP research experts to generate utterances and API calls that require multiple methods from different APIs.

**Crowdsourcing** approaches rely on a group of non-expert contributors to generate datasets, often recruited from platforms like Amazon's Mechanical Turk[17] or Toloka[18]. It enables the collection of a wide range of data by distributing tasks to multiple contributors, who follow instructions for generating or evaluating data [18]. Compared to expert-driven solutions, crowdsourcing is more cost-effective and scalable, as tasks can be distributed to a larger workforce [65]. However, quality assurance is crucial, as non-experts may lack the necessary knowledge to generate accurate data. Common issues include cheating, misspellings, grammatical errors, and lack of semantic relevance, which can compromise the dataset's quality [118].

We found a few datasets that used crowd workers to generate or evaluate dataset quality [19, 29, 110]. PPTC [29] engaged six skilled crowd workers to collect user utterances. MGToolBench [110] hired three crowd workers to evaluate the quality of the generated data. Santos et al. [19] hired crowd workers to generate user utterances related to API methods, encouraging them to introduce variations in grammatical structures and parameter values to enhance linguistic diversity. The generated data was verified by experts to ensure alignment with intended standards. Although we focused on datasets for connecting LLMs to APIs, other related work in the general area of natural language interfaces for digital assistants (e.g., AI assistants, chatbots, conversational agents, etc) leverage crowdsourcing to build training conversational datasets [48, 122].

*5.3.2 Automatic.* This role involves the systematic generation of datasets without human intervention, relying entirely on automated processes. Based on our analysis, this method can be categorized into two approaches: LLM-based and rule-based.

[17] https://www.mturk.com/
[18] https://toloka.ai/

**LLM-based** methods use LLMs to generate synthetic data using prompts. This process often relies on three main components to guide data generation: the specification of the task, the definition of conditions, and the inclusion of in-context examples [57]. First, the task is specified by outlining the purpose and scope of the data to be generated. Second, conditions, guidelines, and constraints are defined to maintain the quality of the text. Finally, in-context examples are included to demonstrate the desired format, offering the model references to enhance alignment with the intended output.

We found that many works use LLMs to curate datasets using different strategies [9, 14, 21, 22, 25, 31, 41, 56, 64, 66, 67, 73, 90, 100, 104, 109, 110, 113, 131]. Some datasets rely on a single curation pattern or generate utterances and API calls using a single prompt. Gorilla [67] used GPT-4 to generate ten instruction-API pairs per API, providing API documentation and three manually created in-context examples. MGToolBench [110] used GPT-4 to paraphrase utterances from the ToolBench [73] dataset, aiming to generate more natural utterances. Sheng et al. [94] generated API calls and user utterances within a single prompt using OAS as input. TinyAgent [22] generated user utterances and associated API calls by providing GPT-4 with descriptions of functions.

Other datasets use multiple prompts to generate utterances with different levels of complexity or employ multiple curation patterns using LLMs. ToolBench [73] used three prompts to generate utterances with varying complexity levels by adjusting the number of APIs provided in each prompt, creating utterances that range from single-tool to multi-tool use. API-Pack [31] prompted Mistral-7B in two stages: (1) generating utterances based on API method descriptions and in-context examples; and (2) classifying the generated utterances against author-defined quality guidelines, filtering out low-quality instances. Santos et al. [19] prompted GPT-3.5 to initially create utterances based on OAS and then paraphrase unique utterances to increase diversity.

Alternatively, some datasets employ a multi-stage pipeline that generates datasets in gradual steps. RESTful-LLaMA [113] generated data using three sequential prompts: (1) producing Python code based on API documentation; (2) generating utterances given the generated code; and (3) refining utterances to include the required API method parameters. API-Bank [52] used five specialized agents, each responsible for generating a distinct component: domain, API documentation, user utterance, API call, and response. ToolAlpaca [100] used GPT-3.5 to simulate three virtual agents that interact over multiple rounds to generate user utterances, API calls, and final responses.

**Rule-based** methods systematically curate datasets using predefined rules, logic, or heuristics without relying on humans or LLMs. We found that rule-based methods are used in two main ways: applying automated validation to filter API-augmented data or using structured transformation rules to transform datasets from other domains into API-augmented data.

Automated validation steps filter instances that fail to meet predefined quality standards or sanity checks, including formatting inconsistencies (e.g., output not represented in JSON) [33, 35, 51, 56], argument parsing errors (e.g., missing required parameters or including unknown input parameters and functions names) [22, 45, 56], API call execution failures (e.g., returning 4xx errors) [51, 56], and duplicates or semantically irrelevant utterances using similarity metrics [19, 127, 133]. For instance, APIGen [56] removed API calls that do not conform to JSON format consistency and those that return execution errors. Seal [45] filtered API calls from ToolBench [73] that either return errors due to argument parsing issues or contain hallucinated API methods. OOD-Toolset [35] programmatically removed instances from ToolAlpaca [100] based on four criteria: missing keys or formatting errors; utterances requiring more than three API calls; parameters unrelated to the core functionality of APIs (e.g, API keys and sensitive user information); and APIs with fewer than three utterance-API calls. ToolBH [133] and Santos et al. [19] used embedding models to filter paraphrased utterances based on the cosine similarity score between each paraphrase and the original seed. While ToolBH [133] eliminates paraphrases that are too similar to the original utterance, Santos et al. [19] filter those with low similarity scores to ensure the paraphrases are

semantically related to the original utterance. RoTBench [127] removes duplicates by filtering generated utterances using Rouge-L similarity rather than embedding models, discarding those with a Rouge-L score exceeding 0.55 compared to seed utterances.

Structured transformation rules programmatically transform datasets focused on other domains (e.g., personal assistants or task-oriented datasets) into API-augmented data [5, 63, 104]. For example, API-Blend [5] generated API calls by processing a dataset of multi-intent utterances created with "and" statements that lacked API calls. They first split the utterances into their original single-intent forms, then parsed parameters to obtain their names and values, and finally merged the utterances back, now enriched with API calls. NLSI [63] generated API calls by repurposing a task-oriented dataset containing conversations between a human and a virtual assistant. The process involved extracting slot-value pairs from dataset annotations and mapping them to API parameters using predefined rules. These parameters were then used to construct API calls. AppBench [104] generated API calls by programmatically parsing system turns in multi-turn dialogue datasets. The process involves extracting domain names and API identifiers and establishing dependencies between API arguments based on the sequence in which API calls should be made to fulfill the utterance.

*5.3.3 Hybrid.* This role integrates automated techniques with human expertise to curate API-augmented datasets, addressing limitations of scalability related to human curation and quality related to automated approaches. Based on our analysis, this method can be categorized into two strategies: LLM-human collaboration and template-based methods.

In **LLM-human collaboration**, LLMs generate utterances and human reviewers filter or refine them. We found that many work follow this approach [4, 12, 33, 38, 39, 83, 92, 101, 111, 119, 123, 130, 132, 133]. ToolE [39] generated utterances using GPT-4, and human experts manually filtered utterances that do not comply with predefined rules (e.g., removing repetitive utterances or those containing API names). ToolVerifier [61] prompted LLaMA-2 70B to generate three utterances per API, which were manually reviewed for accuracy. T-Eval [12] created utterances using GPT-3.5 and then annotated API calls using a multi-agent LLM-based system, followed by human reviewers who manually removed incorrect utterance-API call pairs. NestFul [4] prompted Mixtral to generate utterances and applied a two-step filtering process to refine the dataset. They first used rule-based algorithms to validate instances (e.g., ensuring required parameters are included), then manually reviewed and removed examples with semantic errors or incorrect variable assignments. ToolLens [74] used four sequential prompts through GPT-4 to generate user utterances that require multiple APIs, with human verification at each step to ensure quality.

In **template-based** approaches, humans or LLMs design templates of utterances with placeholders for parameters. These placeholders are dynamically filled using rule-based programs, generating utterances with different parameter values while maintaining a consistent structure. We found that some work follows this approach to generate utterances [51, 103, 114]. ToolBench2 [114] manually created templates for utterances and API calls, including placeholders that map keywords in the utterance to arguments in the API calls. For each keyword, they defined a pool of values and systematically generated variations by randomly selecting values to fill the placeholders. WorkBench [99] features manually designed templates that capture workplace activities, such as scheduling meetings, managing emails, and handling project tasks. These templates were expanded to generate utterances by adjusting parameters such as participants, time frames, and contextual details.

## 5.4 Quality Attributes

Quality attributes define the criteria used to evaluate and curate API-augmented datasets and benchmarks, influencing how data is transformed, paraphrased, or filtered to improve the dataset quality. Ensuring high-quality data impacts the performance in text-based applications, leading to

improvements across NLP tasks [89]. Based on our analysis, we found four quality attributes used to curate datasets and benchmarks: diversity, naturalness, completeness, and fluency.

*5.4.1 Diversity.* This attribute refers to the different wordings, expressions, and sentence structures of user utterances. It reflects the richness of language, where individuals express the same intent in multiple ways based on their background and writing style [116]. The diversity of user utterances can be categorised in three ways: lexical, involving the use of a wide range of words (e.g., "*show*" can be replaced by "*display*" or "*present*"); syntactical, referring to using different syntax structures (e.g., "*Can you find beginner-friendly workouts?*" is a question and "*Find workouts that are beginner-friendly* is a command"); and semantics, representing different writing styles that conveys the same intent but emphasizes different aspects, such as distinct tones (e.g., informal vs. formal), sentiment (e.g., happy, sad, or angry), cultural background, or quality-of-services [28, 102].

We found that many works aim to increase diversity at different levels [11, 19, 39, 58, 63, 94, 100, 103, 111, 113, 131]. Santos et al. [19] increased lexical diversity by paraphrasing utterances while penalizing overused words. AppWorld [103] and Sheng et al. [94] focused on parameter diversity, generating utterances with rich parameter values or combinations. AppWorld generates utterances by replacing placeholders with different parameter values, and Sheng et al. [94] generated utterances by changing both the number of parameters used and their values. Re-Invoke [11] introduced controlled diversity by changing temperature values in LLMs to generate diverse utterances. NLSI [63] covered lexical, syntactical, and semantic diversity, generating paraphrases with different wording, sentence structures, and tone shifts. ToolAlpaca [100] emphasized the generation of utterances with lexical diversity by avoiding verb repetition across utterances and syntactical diversity by using a mix of structures (e.g., interrogative sentences, first-person statements, imperative sentences). ToolE [39] introduced semantics and syntactical diversity, generating utterances with different emotional tones (e.g., happiness, anger, excitement) and sentence forms (e.g., questions, commands). RESTful-LLaMA [113] expanded semantics diversity, prompting LLMs to consider factors such as gender (e.g., male, female), mood (e.g., happy, angry), English dialect (e.g., American, British, Australian), and age group (e.g., teen, adult, senior).

*5.4.2 Naturalness.* This attribute measures how closely user utterances resemble real-world queries commonly used by users while avoiding artificial patterns [117]. Given the abstract nature of naturalness, defining how natural an utterance is can be challenging, especially since people with different backgrounds may phrase things differently. However, we show some examples taken from real datasets that are unnatural in Table 1. Example 1 is overly verbose and combines two unrelated requests (finding properties and retrieving articles) into a single utterance. This structure is unnatural, as users usually separate distinct tasks into different messages. Example 2 explicitly mentions the specific API (Movies-Game of Thrones Quotes-API), which is unlikely in a real utterance. Users generally describe their needs (e.g., "Get a Game of Thrones quote") rather than mentioning the API name. Example 3 includes uncommon parameters that users wouldn't typically provide, such as precise latitude and longitude coordinates. Instead, a more natural utterance would reference a city name or landmark.

We found that some work focused on evaluating or developing natural utterances [3, 64, 67, 90, 92, 103, 110, 111, 131]. TaskBench [92] evaluated the naturalness of utterances by randomly sampling 50 examples and requesting human annotators to rate them on a scale from 1 to 5. MGToolBench [110] paraphrases utterances from ToolBench [73] to create more natural utterances by reducing the inclusion of overly specific details (e.g., removing mentions to APIs, methods, and domains). ToolBank [64] enhanced naturalness by randomly sampling ten API methods and prompting an LLM to generate utterances using 2 to 5 methods, ensuring the selection of more contextually aligned API. FlowBench [111], ALMITA [3], and Ask-Before-Plan [131] deliberately added casual

Table 1. Examples of unnatural utterances from existing datasets

| # | Utterance | Dataset |
|---|---|---|
| 1 | As a researcher studying sustainable energy technologies, I need to find properties in Berlin and review no more than three of these properties. Moreover, I need to find articles on Arxiv related to 'solar energy' and get the meta information for up to three of these articles. | T-Eval [12] |
| 2 | How do I use the Movies-Game Of Thrones Quotes-API to obtain a random quote from the series? | API Pack [31] |
| 3 | I am currently reading travel blogs from the US and I am interested in exploring the state of Minnesota (US-MN) with the coordinates 37.4666405 latitude and -85.89465 longitude. | ToolLens [74] |

chit-chat elements into utterances to simulate natural interactions where users include off-topic text. For instance, the utterance "*Hey, I was just thinking about my last vacation! Can you tell me the weather forecast for Tokyo next weekend?*" includes an unrelated personal remark that users can write even when interacting with chatbots.

*5.4.3 Completeness.* This attribute ensures that user utterances contain all essential information (e.g., required parameters) necessary to make a functional API call, whether in single or multiple turns. It also emphasizes clarity, avoiding ambiguous phrasing that could lead to multiple interpretations. For instance, the utterance "*Book a flight for me.*" is too vague and lacks essential details such as the departure city, destination, date, and possibly other required parameters.

Some datasets evaluate completeness or make additional efforts to ensure that even optional parameters and constraints are included in utterances [38, 100, 112]. ToolAlpaca [100] assesses whether an utterance contains all the required information for an API call. UltraTool [38] extends this evaluation to plans, checking whether all user requests are addressed in the execution plan. TravelPlanner [112] and Gorilla [67] integrate user-need constraints to enhance utterance completeness and personalization. TravelPlanner [112] considers budget, flight preferences, and accommodation type within the travel domain, while Gorilla [67] incorporates constraints like parameter size and minimum accuracy for ML models. The constraints ensure that utterances are not only complete but also enriched with optional or personalized preferences.

In contrast, other datasets remove required information from utterances to evaluate whether LLMs can recognize missing conditions or to train LLMs to select correct APIs even with incomplete inputs [19, 58, 62, 107, 131]. For instance, Ask-Before-Plan [131] generates under-specified utterances by removing details about booking flights (e.g., origin, destination, budget). AgentInstruct [62] creates utterances with missing required parameters or unnecessary superfluous ones.

*5.4.4 Fluency.* This attribute refers to grammatical correctness of utterances, ensuring the dataset maintains a high linguistic standard [7]. We found that few studies evaluate fluency [38, 64, 104, 110]. ToolBank [64] compares the fluency of its generated utterances with those from ToolBench [73]. AppBench [104] assesses fluency by considering factors such as clarity, coherence, and readability. In contrast, APIGen [56] intentionally introduces misspellings in user utterances to train LLMs to recognize API intent despite user typos. Fluency is often overlooked in datasets generated by LLMs, given their proficiency in producing fluent text [41]. Some studies refer to this attribute using different terms, such as "*plausibility*", "*syntactic soundness*", and "*language correctness*".

## 6 RQ3. DATASETS CHARACTERISTICS

In this section, we explore fundamental properties of datasets designed to connect LLMs with APIs. We identified four main aspects: scale, domain, user-LLM interaction mode, and APIs composition.

### 6.1 Scale

We analyze the scale of API-augmented datasets in two ways: the number of instances and the number of APIs.

*6.1.1 Number of instances.* It refers to the total number of instances in the curated dataset. In general, datasets with a larger number of diverse instances tend to capture more nuances, leading to higher performance [6]. We categorize datasets according to the number of instances.

We define **small-scale datasets** as those containing fewer than 1,000 instances [4, 12, 21, 34, 61, 66, 83–85, 95, 98, 99, 103, 104, 107, 111, 123, 127, 128, 133]. They often represent benchmarks for evaluating the API-augmented capabilities of LLMs, and are commonly manually generated or verified to ensure quality. AppWorld [103] provides a dataset with 750 instances manually generated and covering complex scenarios that require planning and robust failure handling. Hao et al. [34] create a small dataset of 39 challenging travel-related utterances featuring unsatisfiable constraints (e.g., non-stop flights, airline restrictions, unavailable attractions). The dataset is designed to assess how LLMs handle scenarios where user-specified constraints cannot all be met. Small-scale datasets often represent high-quality datasets. However, the limited size may restrict their ability to represent diverse real-world interactions.

We define **medium-scale datasets** as those containing between 1,000 and 10,000 instances [3, 33, 35, 38, 52, 58, 63, 100, 101, 112, 114, 119, 130–132]. These datasets are often built using a combination of human and automated efforts, resulting in a good balance between size and quality control. Zhang et al. [132] utilized GPT-4 to generate a dataset comprising 1,550 instances of API documentation, user utterances, and API calls, followed by manual filtering to eliminate incorrect instances. Ultratool [38] combined GPT-4 and experts' efforts to generate a dataset with 5,824 instances. Developers first created user utterances, then GPT-4 generated diverse utterances and API calls, and finally, six experts manually reviewed the GPT-generated data.

We define **large-scale datasets** as those containing more than 10,000 instances [2, 5, 9, 13, 14, 22, 25, 30, 31, 45, 51, 56, 64, 67, 73, 74, 83, 90, 92, 109, 110, 113]. These datasets are mainly generated using LLMs, as manually generating such large datasets is costly and time-consuming. RESTful-LLaMA [113] used GPT-4o to generate a dataset with 29,968 instances. API-Pack [31] used Mistral-7B to generate 1,128,599 instances of API calls across multiple programming languages, including Python, cURL, Java, Node.js, and JavaScript. Large-scale datasets are crucial for building general-purpose models and are typically designed to enhance diversity and generalization. Challenges include quality control and the risk of including repetitive examples. However, to mitigate quality concerns, some work uses LLMs to assess quality and filter poor instances [9, 31, 41].

*6.1.2 Number of APIs.* It refers to the number of APIs used in the dataset. Datasets with a broader selection of APIs enable models to learn patterns applicable to a wider range of tasks [40]. We categorize datasets according to the number of APIs.

We define **small API coverage** datasets as those with 100 APIs or fewer [4, 12, 22, 51, 58, 63, 64, 66, 85, 92, 95, 98, 99, 103, 104, 107, 112, 123, 127, 131]. Although some focus on specific areas, such as programming [64], or customer service [123], most focus on a few different tasks covering multiple domains. For example, TARA [51] includes seven APIs covering tasks such as translation, search, weather updates, and calendar management. In most cases, these datasets typically have

fewer instances and are easier to manually evaluate for quality. Also, the limited number of APIs simplifies the process of selecting API methods.

We define **large API coverage** datasets as those with more than 100 APIs [2, 9, 21, 25, 31, 33, 35, 38, 39, 45, 52, 56, 61, 73, 74, 83, 90, 100, 103, 109, 113, 114, 132]. These datasets often address multiple domains. For instance, ToolBench [73] and UltraTool [38] include information about 16,464 and 2,032 APIs, respectively. In contrast, FinVerse [2] includes 642 APIs exclusively in the finance domain. Although datasets with larger API coverage provide broader diversity and generalisability, their size makes quality verification more challenging. In addition, a higher number of APIs makes it more challenging for an API retriever to select relevant APIs.

### 6.2 Domain

Datasets can also be categorized based on the domains they cover. We classify datasets as either domain-specific or general-purpose.

*6.2.1 Domain-specific datasets.* These datasets focus on a particular area. Examples include PowerPoint automation [135], machine learning models [67], finance [2], programming [64, 138], entertainment (movie and music) [98], customer service [3, 123], edge computing [22], cloud computing [37], home assistance [79], travelling [34, 112, 131], or workplace setting (e.g., sending emails, scheduling events) [99]. In addition, one interesting application is the development of benchmarks focusing on safety issues when connecting LLMs with APIs [14, 83, 84, 130]. For instance, ToolEmu [83] provides a benchmark that contains user utterances with situations where incorrect execution could lead to significant risks, such as deleting entire folders or sending money to the wrong recipient. Similarly, InjectAgent [130] evaluates the vulnerability of API-augmented LLMs to indirect prompt injection, where malicious instructions are embedded within the content processed by the LLMs to manipulate them into executing harmful actions.

*6.2.2 General-purpose datasets.* These datasets include utterances covering multiple domains. We found that most reviewed datasets fall into this category [4, 5, 9, 12, 13, 21, 23, 25, 30, 31, 33, 35, 38, 39, 45, 51, 52, 56, 58, 61–63, 66, 73, 74, 90, 92, 100, 101, 103, 104, 109–111, 113–115, 119, 127, 128, 132]. A key challenge is maintaining both quality and diversity across domains while minimizing bias toward specific applications. For instance, APIGen [56] features instances from 21 categories, covering areas like travel, music, and sports. On the other hand, FlowBench [111] covers only six domains, including customer service, travel and transportation, and personal assistance.

### 6.3 User-LLM interaction mode

We also classify datasets based on the number of message turns exchanged between users and LLMs. We found that it can be categorized into two modes: single-turn and multi-turn conversations.

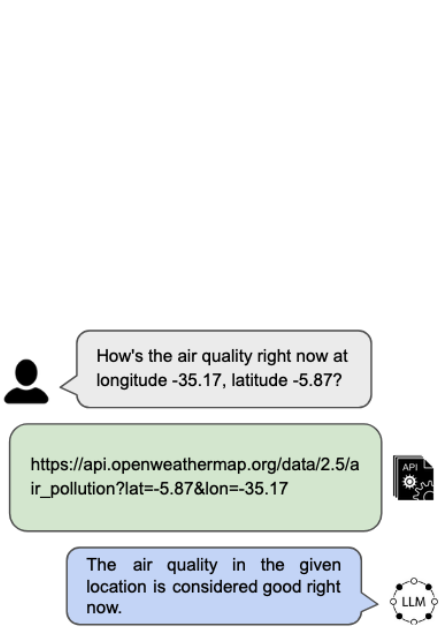
*6.3.1 Single-turn conversation.* There is only one turn of messages exchanged between the user and the LLM. Figure 7a illustrates an example adapted from the ToolBench2 [114] dataset, where the user requests information about air quality at a given location. The LLM obtains weather information after invoking an API and responds to the user that the air quality is good, concluding the interaction in a single turn. We found that most datasets reviewed in this study adopt this mode [4, 5, 9, 12, 14, 21, 22, 25, 30, 31, 33, 35, 38, 39, 51, 56, 61, 64, 67, 73, 74, 83, 85, 92, 98, 99, 101, 104, 109, 110, 112–115, 127, 128, 130, 132].

There are two limitations to the naturalness of datasets that use a single-turn interaction mode. First, user utterances must contain all necessary information within a single message, since there are no additional turns for clarification. This can make interactions feel unnatural compared to real-world conversations, where users provide information gradually over multiple exchanges

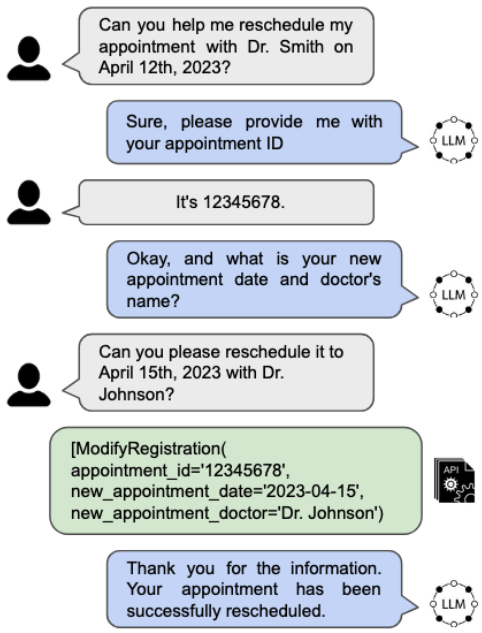
[129]. Second, single-turn utterances requiring multiple APIs from different domains can also feel unnatural. For example, it is uncommon for a user to ask about the weather, currency exchange rates, and sports scores in a single utterance, as seen in the first example in Table 1. Again, in real-world scenarios, such utterances are typically handled through multiple turns, allowing users to engage in different topics more naturally [129].

*6.3.2 Multi-turn conversation.* Multiple rounds of messages are exchanged between the user and the LLM. Figure 7b illustrates an example adapted from the API-Bank [52] dataset, where messages are exchanged across three turns before an API call is made to reschedule a medical appointment. In this example, all the required information (e.g., appointment ID, new appointment date, and doctor name) is provided gradually by the user, representing a more natural interaction.

We found that some datasets reviewed in this study adopt this mode [3, 23, 52, 58, 63, 100, 107, 111, 119, 123, 131, 135]. These datasets mostly include user utterances with missing information in the first message exchanged with the LLM. In such scenarios, the LLM must identify the missing information, request it from the user or retrieve it based on previous messages, and incorporate it in the API call to fulfill the utterance. Some datasets also include utterances with ambiguity (e.g., "*schedule a meeting with Alex*", which is unclear if multiple contacts named Alex exist) or incorrect information (e.g., wrong login details), requiring the LLM to ask for clarification.




(a) Example of a single-turn interaction mode. Example adapted from the ToolBench2 [114] dataset.




(b) Example of a multi-turn interaction. Example adapted from the API-Bank [52] dataset.

Fig. 7. Instances highlighting the single-turn and multi-turn interactions between users and LLMs.

Although multi-turn interactions better reflect real-world scenarios, building such datasets is challenging since it requires generating conversations between the LLM and the user. Most datasets serve as benchmarks for testing multi-turn interactions and are either entirely manually built [15, 58, 107] or manually refined after using LLMs to assist in creating conversations [3, 23, 111, 123].

These datasets are small in scale, given the manual effort to build them. For instance, NoisyToolBench [107], ALMITA [3], and ToolSandbox [58] contain 200, 192, and 224 instances with multi-turn conversations, respectively. The generation of larger multi-turn datasets commonly involves the use of multi-agent LLM-based frameworks [52, 100, 131]. For instance, ToolAlpaca [100] generates 3,938 instances using three GPT-based agents. API-Bank [52] generates 1,888 dialogues using five LLM-based agents. Since manually verifying these datasets is expensive, researchers assess the dataset by fine-tuning LLMs on the generated data and comparing their performance with models not trained in such datasets [52, 100]. The intuition is that datasets have high quality if the fine-tuned model performs better.

### 6.4 APIs composition

The APIs composition refers to the number of API calls needed to fulfill the user's utterance. We classify API composition into two categories: single and multiple API calls.

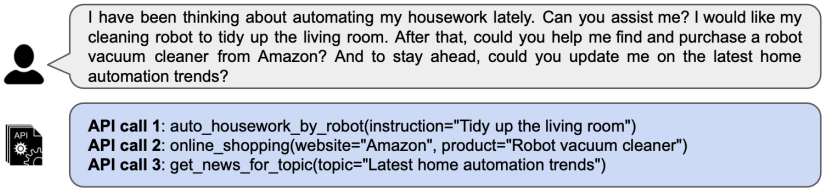
*6.4.1 Single API call.* This composition is used when a user utterance can be fulfilled using one API call. Figure 4 illustrates an example where one API call is made to answer the utterance requesting the weather forecast in New York City for the next five days. We found that most datasets include utterances that are fulfilled using a single API call [9, 19, 31, 35, 38, 39, 45, 51, 56, 58, 61, 62, 67, 83, 92, 95, 99–101, 104, 109, 114, 128, 131]. Furthermore, when generating datasets with single API call instances, authors commonly provide the curator with an API method description and request the generation of utterances that can be fulfilled using that method.

*6.4.2 Multiple API calls.* This composition is necessary for scenarios where fulfilling a user utterance requires coordinating multiple API calls. These API calls may belong to the same API or span across multiple APIs from different domains. Moreover, we found that utterances requiring multiple APIs can be further categorized into two types: independent and nested calls.

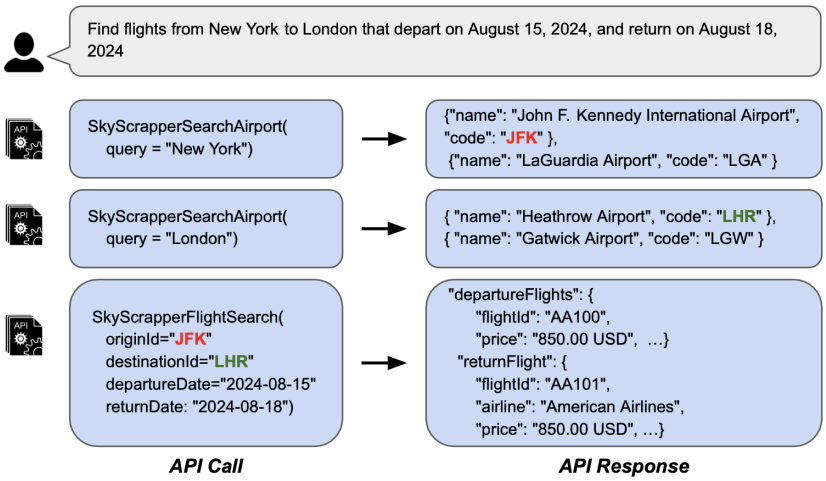
In **independent calls**, the input and output of consecutive calls are unrelated, allowing each call to be executed sequentially without exchanging parameters. Figure 8a provides an example adapted from the TaskBench [92], where three independent calls are made: the first instructs a robot to clean the room; the second retrieves product information from Amazon; and the third fetches the latest news about home automation trends. The order of API calls determines which actions start first, but there are no parameter dependencies between them to fulfill the utterance.

We found that most datasets reviewed in this study include instances with independent calls [5, 12, 21, 23, 25, 30, 39, 45, 51, 52, 56, 62–64, 73, 74, 83, 90, 92, 99, 103, 104, 110–112, 115, 123, 127, 128]. These calls may involve a single API, multiple APIs within the same domain, or multiple APIs across different domains. API-Bank [52] includes multiple independent calls related to a single API. API-Blend [5] incorporates API calls from multiple APIs within the same domain, while AppBench [104] includes API calls from multiple APIs spanning different domains.

In **nested calls**, the output of one API call serves as an input parameter for a subsequent call [38]. Figure 8b shows an example adapted from the NESTful [4] dataset, where a user requests flight information from New York to London. The LLM retrieves the airport codes for the mentioned cities (JFK for New York and LHR for London) and uses these codes to request flight details. Once the flight information is obtained, the LLM presents it to the user in a structured format, listing available flights, their prices, and departure times. Nested API calls are common in scenarios where intermediate results are required to complete a user utterance. For instance, one typical case involves using an API method to retrieve an identifier (ID), which is then passed to another method to obtain information or fulfill a task. Although nested composition closely reflects real-world scenarios, datasets supporting nested calls are more challenging to construct and evaluate, as each step in the sequence should be validated to ensure correctness [36]. Additionally, if an initial API




(a) Example of independent multiple API calls used to address the user utterance. Example taken from the TaskBench [92] dataset.




(b) Example of nested multiple API calls used to address the user utterance. Example taken from the NESTful [4] dataset.

Fig. 8. Instances illustrating the independent and nested approaches for linking user utterances to multiple APIs.

call fails or returns an error, all subsequent calls relying on its output may also fail. To mitigate this, robust error-handling strategies are essential to maintain system reliability [4].

We found that some datasets reviewed in this study include instances with nested calls [4, 22, 33, 38, 85, 95, 98, 104, 109, 132]. NESTFUL [4] manually generated a benchmark with 300 nested-tool calling instances. These instances assess an LLM's ability to identify dependencies, retrieve intermediate outputs, and correctly structure API calls to fulfill complex utterances. GraphQLRestBench [85] generated 155 nested API calls using utterances and API documentation from RestBench [98].

## 7 DISCUSSION

In this section, we provide a summary of our findings and discuss current challenges, future research directions, and potential risks associated with the SLR.

### 7.1 Summary of findings

We discuss the findings related to the three dimensions covered in this paper: LLM-API integration tasks, dataset curation, and dataset characteristics.

*7.1.1 LLM-API Integration Task.* We found that most datasets and benchmarks focus on two tasks: API selection and API calling. API selection involves identifying the most suitable APIs to satisfy a given user utterance. This task is typically addressed using embedding models to retrieve relevant APIs from a large collection and then providing these APIs as input to LLMs. API calling maps parameters from a user utterance to the arguments of an API method and calls the API method to obtain the necessary information or execute an action.

Also, fewer works introduce datasets for planning and API awareness. Planning refers to the decomposition of complex user utterances into a sequence of steps. Tool awareness evaluates the LLM's ability to determine whether a user utterance can be answered using only its internal knowledge or requires APIs.

*7.1.2 Dataset curation.* Table 2 presents an overview of the curation process of some datasets and benchmarks. It shows information about the source of the API documentation used to curate the data, the curation pattern (e.g., transforming, filtering, or paraphrasing), and the curator role (e.g., human, LLMs, or rule-based).

Table 2. Overview of the data curation strategies used in existing datasets and benchmarks for LLM-API integration. The **API Source** column indicates the source of the API documentation used to curate data. The **Curation Pattern** column identifies the data pattern, which can be ⓣransforming, ⓟaraphrasing, or ⓕiltering, as described in Section 5.2. The **Curation Role** column represents the role responsible for data curation: ① human-generated, ② LLM-generated, or ③ rule-based, as outlined in Section 5.3.

| Dataset | API Source | Curation Pattern | Curation Role |
|---|---|---|---|
| ToolBench [73] | RapidAPI | ⓣ | ② |
| Nestful [4] | RapidAPI | ⓣ ⓕ | ① ② ③ |
| ToolSandBox [58] | RapidAPI | ⓣ ⓟ | ① |
| RestBench [98] | Public APIs | ⓣ | ① |
| ToolBench2 [114] | Public APIs | ⓣ | ① ③ |
| API-Pack [31] | Public APIs | ⓣ ⓕ | ② |
| Santos et al. [19] | Public APIs | ⓣ ⓟ ⓕ | ② ③ |
| ToolE [39] | OpenAI plugins | ⓣ ⓟ ⓕ | ① ② |
| API-Blend [5] | Existing datasets | ⓣ | ② ③ |
| MGToolBench [110] | Existing dataset [73] | ⓟ | ② |
| NoisyToolBench [107] | Existing dataset [73] | ⓟ | ① |
| APIGen [56] | Existing dataset [73] | ⓣ ⓕ | ② ③ |
| NLSI [63] | Existing dataset [77] | ⓣ ⓟ | ② ③ |
| OOD-Toolset [35] | Existing dataset [100] | ⓕ | ① ③ |
| Iskander et al. [41] | Existing datasets [73, 100] | ⓕ | ② |
| API-Bank[52] | LLM-based generation | ⓣ | ② |
| ToolAlpaca [100] | LLM-based generation | ⓣ | ② |
| Seal-tools [109] | LLM-based generation | ⓣ | ② |
| UltraTool [38] | LLM-based generation | ⓣ ⓟ ⓕ | ① ② |
| ToolTalk [23] | Human-based generation | ⓣ | ① ② |
| WorkBench [99] | Human-based generation | ⓣ | ① ③ |
| T-Eval [12] | Human-based generation | ⓣ ⓟ ⓕ | ① ② |

We found that most datasets rely on previous work in the field to obtain API documentation. These include online repositories (e.g., RapidAPI, OpenAI plugins, and Public APIs available on

official websites) or existing datasets. ToolBench [73] is an example of a widely used dataset. Fewer works use API documentation generated manually or using LLMs. However, manual creation is time-consuming, and LLM-generated documentation may require additional verification.

Most work curates datasets using the transforming pattern, which modifies, adapts, or generates utterances or API calls based on OpenAPI specifications or datasets from other domains. The second most commonly used pattern is paraphrasing, which rewrites or restructures user utterances to linguistic diversity. Lastly, some work uses the filtering pattern, which removes utterances or API calls that do not meet predefined quality attributes.

We also found that LLMs are mostly used for dataset curation, generating large-scale datasets at a relatively low cost and short time [6, 50, 53]. However, the outputs of LLMs may suffer from inaccuracies, bias, or hallucinations [78]. To mitigate these issues, many works incorporate human reviewers to filter or paraphrase LLM-generated utterances to enhance the dataset quality. Alternatively, some work uses rule-based methods to generate or filter instances, although the resulting data quality is still more limited compared to approaches involving humans.

As illustrated in Table 2, we found that many works combine multiple curation patterns and roles to improve the dataset quality. For instance, WorkBench [99] manually generates templates of user utterances with parameter placeholders, which are filled using rule-based algorithms. T-Eval [12] generates utterances and paraphrases using an LLM followed by manual filtering. Nestful [4] uses an LLM to generate utterances, followed by rule-based filtering and then manual filtering.

Finally, we found that some work incorporates quality attributes to guide the generation of datasets and benchmarks. Diversity is the most common attribute considered, which generates utterances that cover different ways of expressing the same intent [76]. Other work focuses on generating natural utterances, mimicking how users normally talk and avoiding artificial patterns. Fewer works focus on completeness, which aims to include all required parameters in the utterance, and fluency, which creates grammatically correct text.

*7.1.3 Dataset characteristics.* Table 3 presents an overview of the characteristics of some datasets and benchmarks. It shows the number of instances, number of APIs, target domain, user–LLM interaction mode, and API composition.

We found that most benchmarks contain fewer than 1,000 instances and are manually generated or verified to ensure high quality. In contrast, most datasets used for fine-tuning LLMs include more than 10,000 instances and are often curated using LLMs, as manual generation or validation at this scale is expensive. In addition, most datasets cover multiple domains, while a few focus on particular areas, such as travel, machine learning, or workplace-related tasks.

Most datasets represent dialogues between users and LLMs using a single turn, and often map user utterances to multiple independent API calls. Datasets that support multiple independent calls generally also include instances of utterances mapped to a single API call. Only a small number of datasets explicitly include examples for nested API flows.

## 7.2 Challenges

We found six challenges associated with the development of datasets and benchmarks for integrating LLMs with APIs.

*7.2.1 Limited Dataset Quality and Diversity.* A challenge in existing datasets is the lack of naturalness and diversity in user utterances. Many datasets contain artificially generated utterances that do not reflect how users naturally interact with chatbots. Some studies enforce the generation of single utterances that use multiple APIs, resulting in unnatural utterances that do not accurately represent typical interactions. In addition, datasets often fail to capture lexical, syntactic, and semantic diversity simultaneously. While some efforts have been made to address diversity, they

Table 3. Overview of the characteristics of existing datasets and benchmarks for LLM-API integration. The **Interaction** column indicates whether the dataset includes single-turn or multi-turn dialogues. The **API flow** column distinguishes between single calls, multiple ⓘndependent calls, and multiple ⓝested calls.

| Dataset | # Instances | # APIs | Domain | Interaction | API flow |
|---|---|---|---|---|---|
| ToolBench [73] | 126,486 | 16,464 | General | Single | Multiple ⓘ |
| API-Bank[52] | 1,888 | 2,138 | General | Multiple | Multiple ⓘ |
| ToolAlpaca [100] | 3,938 | 426 | General | Multiple | Single |
| API-Blend [5] | 189,040 | - | General | Single | Multiple ⓘ |
| API-Pack [31] | 1,128,599 | 11,213 | General | Single | Single |
| Seal-tools [109] | 14,076 | 4,076 | General | Single | Multiple ⓘ ⓝ |
| MGToolBench [110] | 17,740 | - | General | Single | Multiple ⓘ |
| StableToolBench [30] | 164,980 | - | General | Single | Multiple ⓘ |
| UltraTool [38] | 5,824 | 2,032 | General | Single | Multiple ⓘ ⓝ |
| APIGen [56] | 60,000 | 3,673 | General | Single | Multiple ⓘ |
| NLSI [63] | 2,040 | 17 | General | Multiple | Multiple ⓘ |
| Nestful [4] | 300 | 39 | General | Single | Multiple ⓝ |
| T-Eval [12] | 533 | 15 | General | Single | Multiple ⓘ |
| ToolE [39] | 21,127 | 390 | General | Single | Multiple ⓘ |
| ToolTalk [23] | 78 | 7 | General | Multiple | Multiple ⓘ |
| APIBench [67] | 16,450 | 1,645 | ML | Single | Single |
| TinyBench [22] | 82,000 | 16 | Edge | Single | Multiple ⓝ |
| RestBench [98] | 157 | 2 | Movie/Music | Single | Multiple ⓘ |
| WorkBench [99] | 690 | 5 | Workplace | Single | Multiple ⓘ |
| TravelPlanner [112] | 1,225 | 7 | Travelling | Single | Multiple ⓘ |

tend to focus on isolated aspects of diversity. Furthermore, we found a few datasets that incorporate quality-of-service (QoS) attributes, which represent utterances that explicitly include qualitative preferences. For instance, when purchasing a refrigerator, a user might specify factors such as cost or energy efficiency. Expanding datasets to include such diverse variations would enable LLMs to better generalize across different user intents.

*7.2.2 Absence of Multi-Turn Datasets.* Most existing datasets focus on single-turn interactions, where users provide long and complex utterances requesting multiple tasks simultaneously. However, real-world conversations with chatbots typically occur in a multi-turn way, where users refine their requests, seek clarifications, and adapt based on responses. The lack of datasets capturing such interactive dialogues limits the ability of LLMs to model real user interactions effectively.

*7.2.3 Overfitting to Dataset-Specific Representations.* Another challenge is the reliance on test sets generated using the same style or prompt format as the training data. While this approach may yield high-performance metrics, it often leads to overfitting, where models perform well on specific dataset characteristics but fail to generalize to more diverse and real-world utterances. This limitation is problematic in datasets that lack natural variation, making the resulting benchmarks unrepresentative of actual user interactions.

*7.2.4 Limitations of Reference-Based Evaluation.* Existing work evaluates the generated API calls by comparing these calls to a predefined gold standard reference. However, this approach fails to account for cases where API calls from different APIs can achieve the same outcome. Rigid

gold-standard comparisons may incorrectly penalize valid responses simply because they differ from the reference. To mitigate this issue, some works, such as ToolSandbox [58] and AppWorld [103], have adopted milestone-based evaluation methods. These approaches assess whether the LLM agent successfully reaches key milestones in solving user utterances rather than matching a predefined reference output. This flexibility allows for more accurate assessments of model effectiveness.

*7.2.5 Lack of Personalization and Context Awareness.* Most existing API-augmented datasets consist of isolated utterance–API call pairs without accounting for prior conversational context or user-specific preferences. However, real-world chatbots often work over multiple turns and adapt their behavior based on previous interactions, preferences, and history. For instance, an LLM-based assistant may learn that a user prefers a 24-hour time format, selects economy-class flights, or frequently uses a specific food delivery service, and use this information to perform API selection and fill API method parameters more efficiently. Although some work takes an initial step toward addressing this issue [63], the development of memory-aware datasets remains limited across existing datasets and benchmarks.

*7.2.6 Overreliance on Synthetic Datasets.* Most existing datasets are synthetically generated using LLMs, which poses risks such as model collapse [97]. Synthetic datasets may fail to capture the nuanced variability of real human interactions, leading to biases in model training.

### 7.3 Future Directions

In response to the identified challenges, we outline the following future directions to guide research on datasets and benchmarks for LLM–API integration.

*7.3.1 Enhancing the naturalness and diversity of datasets.* Future research includes the development of datasets and benchmarks that reflect how users naturally communicate with chatbots. This includes avoiding artificial phrasing and ensuring a rich diversity of lexical, syntactic, and semantic utterances. Enhancing both naturalness and diversity will enable models to generalize more effectively to real-world scenarios and user behaviors.

*7.3.2 Development of datasets with multi-turn interactions.* Future datasets should emphasize multi-turn dialogues to support the development of chatbots that can adapt to more realistic interaction scenarios where users provide information across multiple turns.

*7.3.3 Implementing cross-dataset evaluations.* Future research should focus on developing benchmarks and evaluating models using diverse test datasets with different structures and writing styles. Such cross-dataset evaluation helps ensure robustness and reduces bias introduced by dataset-specific patterns.

*7.3.4 Performing milestone-based evaluations.* Future evaluation frameworks should adopt milestone-based criteria instead of rigid gold-standard references. This approach allows for greater flexibility in assessing chatbot performance by measuring meaningful progress. It is particularly valuable in cases where multiple valid solutions exist.

*7.3.5 Development of memory- and context-aware datasets.* Future research should include the development of memory- and preference-aware datasets that reflect realistic and personalized interaction scenarios. Like humans, LLMs must learn not only how to use tools but also to adapt their behavior based on user-specific preferences and conversation history.

*7.3.6 Reducing overreliance on LLM-based synthetic datasets.* Future research should explore hybrid dataset construction methods that combine manually curated data with synthetic augmentation.

Crowdsourced or expert-annotated datasets can serve as a high-quality foundation, while LLMs can be employed to introduce controlled diversity and expansion. This approach could lead to more representative and effective training data, ultimately improving model robustness.

### 7.4 Risks

Despite our efforts to conduct a comprehensive systematic literature review, this study is subject to limitations. First, the rapid pace of development of API-augmented datasets and benchmarks presents a challenge for maintaining up-to-date coverage. New datasets and benchmarks are being introduced at a fast pace, and the taxonomy defined in this study may have evolved since the time of our review. Second, relevant studies may have been unintentionally excluded due to the ambiguity and inconsistency in the literature regarding the distinction between tools, APIs, and related interfaces. Although we carefully refined our search queries to retrieve the most relevant papers, studies using alternative terminology may not have been captured.

## 8 CONCLUSION

In this paper, we conducted a systematic literature review of datasets and benchmarks designed to integrate Large Language Models (LLMs) with APIs. We reviewed and analyzed 69 papers across three main dimensions: (1) LLM-API integration tasks, investigating the tasks datasets are designed to address; (2) dataset curation, analyzing the process to build API-augmented datasets; and (3) dataset characteristics, describing properties such as scale, domain coverage, user-LLM interaction modes, and API call structure. We also proposed a novel taxonomy for each of the three dimensions to support a more structured understanding of how current work addresses API-augmented datasets and benchmarks. Finally, we summarised our main findings, described current challenges, and proposed future directions to advance research in API-augmented dataset development and evaluation.

## REFERENCES

[1] Mahdi Aben Ahmed. 2025. From Text to Understanding the Inner Text: LLMs and Translation Accuracy and Fluency. *International Journal of Language and Literary Studies* 7, 2 (2025), 139–156. https://doi.org/10.36892/ijlls.v7i2.2072

[2] Siyu An, Qin Li, Junru Lu, Di Yin, and Xing Sun. 2024. FinVerse: An Autonomous Agent System for Versatile Financial Analysis. *arXiv* (2024). https://doi.org/10.48550/arXiv.2406.06379

[3] Samuel Arcadinho, David Oliveira Aparicio, and Mariana S. C. Almeida. 2024. Automated test generation to evaluate tool-augmented LLMs as conversational AI agents. In *Proceedings of the 2nd GenBench Workshop on Generalisation (Benchmarking) in NLP*. Association for Computational Linguistics, Miami, Florida, USA, 54–68. https://doi.org/10.18653/v1/2024.genbench-1.4

[4] Kinjal Basu, Ibrahim Abdelaziz, Kelsey Bradford, Maxwell Crouse, Kiran Kate, Sadhana Kumaravel, Saurabh Goyal, Asim Munawar, Yara Rizk, Xin Wang, et al. 2024. NESTFUL: A Benchmark for Evaluating LLMs on Nested Sequences of API Calls. *arXiv* (2024). https://doi.org/10.48550/arXiv.2409.03797

[5] Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Vernon Austel, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis Lastras. 2024. API-BLEND: A Comprehensive Corpora for Training and Benchmarking API LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 12859–12870. https://doi.org/10.18653/v1/2024.acl-long.694

[6] Jan Cegin, Jakub Simko, and Peter Brusilovsky. 2023. ChatGPT to Replace Crowdsourcing of Paraphrases for Intent Classification: Higher Diversity and Comparable Model Robustness. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 1889–1905. https://doi.org/10.18653/v1/2023.emnlp-main.117

[7] Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. 2020. Evaluation of text generation: A survey. *arXiv* (2020). https://doi.org/10.48550/arXiv.2006.14799

[8] Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024. Advancing Tool-Augmented Large Language Models: Integrating Insights from Errors in Inference Trees. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., 106555–106581.

[9] Wei Chen, Zhiyuan Li, and Mingyuan Ma. 2025. Octopus: On-device language model for function calling of software APIs. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Albuquerque, New Mexico, 329–339. https://doi.org/10.18653/v1/2025.naacl-industry.27

[10] Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-Resource Domain Adaptation for Compositional Task-Oriented Semantic Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 5090–5100. https://doi.org/10.18653/v1/2020.emnlp-main.413

[11] Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-Invoke: Tool Invocation Rewriting for Zero-Shot Tool Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, Miami, Florida, USA, 4705–4726. https://doi.org/10.18653/v1/2024.findings-emnlp.270

[12] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-Eval: Evaluating the Tool Utilization Capability of Large Language Models Step by Step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 9510–9529. https://doi.org/10.18653/v1/2024.acl-long.515

[13] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing Data and Methods of Effective Agent Tuning for Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 9354–9366. https://doi.org/10.18653/v1/2024.findings-acl.557

[14] Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024. Towards Tool Use Alignment of Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Miami, Florida, USA. https://doi.org/10.18653/v1/2024.emnlp-main.82

[15] Cheng-Han Chiang and Hung-yi Lee. 2023. Can Large Language Models Be an Alternative to Human Evaluations?. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Toronto, Canada, 15607–15631. https://doi.org/10.18653/v1/2023.acl-long.870

[16] Victoria Clarke and Virginia Braun. 2017. Thematic analysis. *The journal of positive psychology* 12, 3 (2017), 297–298.

[17] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. 2024. A complete survey on LLM-based AI chatbots. *arXiv* (2024). https://doi.org/10.48550/arXiv.2406.16937

[18] Florian Daniel, Pavel Kucherbaev, Cinzia Cappiello, Boualem Benatallah, and Mohammad Allahbakhsh. 2018. Quality Control in Crowdsourcing: A Survey of Quality Attributes, Assessment Techniques, and Assurance Actions. *ACM Comput. Surv.* 51, 1, Article 7 (Jan. 2018), 40 pages. https://doi.org/10.1145/3148148

[19] Vitor Gaboardi dos Santos, Boualem Benatallah, Auday Berro, and Silvana Togneri MacMahon. 2024. Diverse Utterances Generation with GPT to Improve Task-Oriented Chatbots and APIs Integration. In *2024 2nd International Conference on Foundation and Large Language Models*. 42–50. https://doi.org/10.1109/FLLM63129.2024.10852454

[20] Vitor Gaboardi dos Santos, Guto Leoni Santos, Theo Lynn, and Boualem Benatallah. 2024. Identifying Citizen-Related Issues from Social Media Using LLM-Based Data Augmentation. In *Advanced Information Systems Engineering*. Springer Nature Switzerland, Cham, 531–546. https://doi.org/10.1007/978-3-031-61057-8_31

[21] Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. AnyTool: self-reflective, hierarchical agents for large-scale API calls. In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org, Article 470, 18 pages.

[22] Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Richard Charles Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2024. TinyAgent: Function Calling at the Edge. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 80–88. https://doi.org/10.18653/v1/2024.emnlp-demo.9

[23] Nicholas Farn and Richard Shin. 2023. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv* (2023). https://doi.org/10.48550/arXiv.2311.10775

[24] Michael Fore, Simranjit Singh, and Dimitrios Stamoulis. 2024. GeckOpt: LLM System Efficiency via Intent-Based Tool Selection. In *Proceedings of the Great Lakes Symposium on VLSI 2024*. Association for Computing Machinery, 353–354. https://doi.org/10.1145/3649476.3658784

[25] Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative Tool Learning from Introspection Feedback by Easy-to-Difficult Curriculum. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 16 (2024), 18030–18038. https://doi.org/10.1609/aaai.v38i16.29759

[26] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv* (2023). https://doi.org/10.48550/arXiv.2312.10997

[27] 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* (2024). https://doi.org/10.48550/arXiv.2402.16696

[28] Yanzhu Guo, Guokan Shang, and Chloé Clavel. 2024. Benchmarking linguistic diversity of large language models. *arXiv* (2024). https://doi.org/10.48550/arXiv.2412.10271

[29] Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Nan Duan. 2024. PPTC Benchmark: Evaluating Large Language Models for PowerPoint Task Completion. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 8682–8701. https://doi.org/10.18653/v1/2024.findings-acl.514

[30] 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. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 11143–11156. https://doi.org/10.18653/v1/2024.findings-acl.664

[31] Zhen Guo, Adriana Meza Soria, Wei Sun, Yikang Shen, and Rameswar Panda. 2024. API Pack: A massive multi-programming language dataset for API call generation. *arXiv* (2024). https://doi.org/10.48550/arXiv.2402.09615

[32] Alon Halevy and Jane Dwivedi-Yu. 2023. Learnings from data integration for augmented language models. *arXiv* (2023). https://doi.org/10.48550/arXiv.2304.04576

[33] Han Han, Tong Zhu, Xiang Zhang, MengSong Wu, Xiong Hao, and Wenliang Chen. 2025. NesTools: A Dataset for Evaluating Nested Tool Learning Abilities of Large Language Models. In *Proceedings of the 31st International Conference on Computational Linguistics*. Association for Computational Linguistics, 9824–9844. https://aclanthology.org/2025.coling-main.657/

[34] Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2024. Large language models can plan your travels rigorously with formal verification tools. *arXiv* (2024). https://doi.org/10.48550/arXiv.2404.11891

[35] Wei He, Shichun Liu, Jun Zhao, Yiwen Ding, Yi Lu, Zhiheng Xi, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Self-Demos: Eliciting Out-of-Demonstration Generalizability in Large Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 3829–3845. https://doi.org/10.18653/v1/2024.findings-naacl.243

[36] Saghar Hosseini, Ahmed Hassan Awadallah, and Yu Su. 2021. Compositional generalization for natural language interfaces to web APIs. *arXiv* (2021). https://doi.org/10.48550/arXiv.2112.05209

[37] 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* (2023). https://doi.org/10.48550/arXiv.2308.00675

[38] 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. In *Findings of the Association for Computational Linguistics*. Association for Computational Linguistics, 4363–4400. https://doi.org/10.18653/v1/2024.findings-acl.259

[39] Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv* (2023). https://doi.org/10.48550/arXiv.2310.03128

[40] Sathish Reddy Indurthi, Wenxuan Zhou, Shamil Chollampatt, Ravi Agrawal, Kaiqiang Song, Lingxiao Zhao, and Chenguang Zhu. 2024. Improving Multilingual Instruction Finetuning via Linguistically Natural and Diverse Datasets. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 2306–2323. https://doi.org/10.18653/v1/2024.findings-emnlp.128

[41] Shadi Iskander, Sofia Tolmach, Ori Shapira, Nachshon Cohen, and Zohar Karnin. 2024. Quality Matters: Evaluating Synthetic Data for Tool-Using LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 4958–4976. https://doi.org/10.18653/v1/2024.emnlp-main.285

[42] Hanlei Jin, Yang Zhang, Dan Meng, Jun Wang, and Jinghua Tan. 2024. A comprehensive survey on process-oriented automatic text summarization with exploration of LLM-based methods. *arXiv* (2024). https://doi.org/10.48550/arXiv.2403.02901

[43] Lucas Joos, Daniel A Keim, and Maximilian T Fischer. 2024. Cutting Through the Clutter: The Potential of LLMs for Efficient Filtration in Systematic Literature Reviews. *arXiv* (2024). https://doi.org/10.48550/arXiv.2407.10652

[44] Mohammad Kachuee, Sarthak Ahuja, Vaibhav Kumar, Puyang Xu, and Xiaohu Liu. 2025. Improving Tool Retrieval by Leveraging Large Language Models for Query Generation. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*. Association for Computational Linguistics, 29–38. https://aclanthology.org/2025.coling-industry.3/

[45] Woojeong Kim, Ashish Jagmohan, and Aditya Vempaty. 2024. SEAL: Suite for Evaluating API-use of LLMs. *arXiv* (2024). https://doi.org/10.48550/arXiv.2409.15523

[46] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466.

https://doi.org/10.1162/tacl_a_00276

[47] Stefan Larson and Kevin Leach. 2022. A survey of intent classification and slot-filling datasets for task-oriented dialog. *arXiv* (2022). https://doi.org/10.48550/arXiv.2207.13211

[48] Stefan Larson, Anish Mahendran, Andrew Lee, Jonathan K. Kummerfeld, Parker Hill, Michael A. Laurenzano, Johann Hauswald, Lingjia Tang, and Jason Mars. 2019. Outlier Detection for Improved Data Quality and Diversity in Dialog Systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 517–527. https://doi.org/10.18653/v1/N19-1051

[49] Patrick Lewis, Barlas Oguz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2020. MLQA: Evaluating Cross-lingual Extractive Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 7315–7330. https://doi.org/10.18653/v1/2020.acl-main.653

[50] Jiyi Li. 2024. A Comparative Study on Annotation Quality of Crowdsourcing and LLM via Label Aggregation. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 6525–6529.

[51] Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2023. Tool-augmented reward modeling. *arXiv* (2023). https://doi.org/10.48550/arXiv.2310.01045

[52] 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*. Association for Computational Linguistics, 3102–3116. https://doi.org/10.18653/v1/2023.emnlp-main.187

[53] Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2024. From Crowdsourced Data to High-Quality Benchmarks: Arena-Hard and BenchBuilder Pipeline. *arXiv* (2024). https://doi.org/10.48550/arXiv.2406.11939

[54] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2024. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing* 3 (2024), 0063. https://doi.org/10.34133/icomputing.0063

[55] Yupian Lin, Tong Ruan, Jingping Liu, and Haofen Wang. 2023. A survey on neural data-to-text generation. *IEEE Transactions on Knowledge and Data Engineering* (2023). https://doi.org/10.1109/TKDE.2023.3304385

[56] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APIGen: Automated PIpeline for Generating Verifiable and Diverse Function-Calling Datasets. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., 54463–54482.

[57] Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On LLMs-Driven Synthetic Data Generation, Curation, and Evaluation: A Survey. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 11065–11082. https://doi.org/10.18653/v1/2024.findings-acl.658

[58] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, 1160–1183. https://doi.org/10.18653/v1/2025.findings-naacl.65

[59] 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. 36 (2023), 43447–43478.

[60] Zexiong Ma, Shengnan An, Bing Xie, and Zeqi Lin. 2024. Compositional API Recommendation for Library-Oriented Code Generation. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. Association for Computing Machinery, 87–98. https://doi.org/10.1145/3643916.3644403

[61] Dheeraj Mekala, Jason E Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. 2024. TOOLVERIFIER: Generalization to New Tools via Self-Verification. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 5026–5041. https://doi.org/10.18653/v1/2024.findings-emnlp.289

[62] Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. 2024. Agentinstruct: Toward generative teaching with agentic flows. *arXiv* (2024). https://doi.org/10.48550/arXiv.2407.03502

[63] Nikita Moghe, Patrick Xia, Jacob Andreas, Jason Eisner, Benjamin Van Durme, and Harsh Jhamtani. 2024. Interpreting User Requests in the Context of Natural Language Standing Instructions. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 4043–4060. https://doi.org/10.18653/v1/2024.findings-naacl.255

[64] Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. 2024. Efficient and scalable estimation of tool representations in vector space. *arXiv* (2024). https://doi.org/10.48550/arXiv.2409.02141

[65] Steven Moore, Norman Bier, and John Stamper. 2024. Assessing Educational Quality: Comparative Analysis of Crowdsourced, Expert, and AI-Driven Rubric Applications. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing* 12, 1 (2024), 115–125. https://doi.org/10.1609/hcomp.v12i1.31606

[66] Honglin Mu, Yang Xu, Yunlong Feng, Xiaofeng Han, Yitong Li, Yutai Hou, and Wanxiang Che. 2024. Beyond Static Evaluation: A Dynamic Approach to Assessing AI Assistants' API Invocation Capabilities. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation*. ELRA and ICCL, 2342–2353. https://aclanthology.org/2024.lrec-main.209/

[67] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., 126544–126565.

[68] Ondřej Plátek, Vojtech Hudecek, Patricia Schmidtova, Mateusz Lango, and Ondrej Dusek. 2023. Three Ways of Using Large Language Models to Evaluate Chat. In *Proceedings of the Eleventh Dialog System Technology Challenge*. Association for Computational Linguistics, 113–122. https://aclanthology.org/2023.dstc-1.14/

[69] Joshua S Ponelat and Lukas L Rosenstock. 2022. *Designing APIs with Swagger and OpenAPI*. Simon and Schuster.

[70] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. ADaPT: As-Needed Decomposition and Planning with Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 4226–4252. https://doi.org/10.18653/v1/2024.findings-naacl.264

[71] Libo Qin, Xiao Xu, Wanxiang Che, and Ting Liu. 2020. AGIF: An Adaptive Graph-Interactive Framework for Joint Multiple Intent Detection and Slot Filling. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, 1807–1816. https://doi.org/10.18653/v1/2020.findings-emnlp.163

[72] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, et al. 2024. Tool Learning with Foundation Models. *ACM Comput. Surv.* 57, 4, Article 101 (2024), 40 pages. https://doi.org/10.1145/3704435

[73] 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* (2023). https://doi.org/10.48550/arXiv.2307.16789

[74] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards Completeness-Oriented Tool Retrieval for Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. Association for Computing Machinery, 1930–1940. https://doi.org/10.1145/3627673.3679847

[75] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-rong Wen. 2025. Tool learning with large language models: a survey. *Front. Comput. Sci.* 19, 8 (2025). https://doi.org/10.1007/s11704-024-40678-2

[76] Jorge Ramírez, Marcos Baez, Auday Berro, Boualem Benatallah, and Fabio Casati. 2022. Crowdsourcing Syntactically Diverse Paraphrases with Diversity-Aware Prompts and Workflows. In *Advanced Information Systems Engineering (CAiSE)*. Springer, 253–269. https://doi.org/10.1007/978-3-031-07472-1_15

[77] Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards Scalable Multi-Domain Conversational Agents: The Schema-Guided Dialogue Dataset. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 05 (2020), 8689–8696. https://doi.org/10.1609/aaai.v34i05.6394

[78] Vipula Rawte, Amit Sheth, and Amitava Das. 2023. A survey of hallucination in large foundation models. *arXiv* (2023). https://doi.org/10.48550/arXiv.2309.05922

[79] Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. 2023. SAGE: Smart home Agent with Grounded Execution. *arXiv* (2023). https://doi.org/10.48550/arXiv.2311.00772

[80] Shamik Roy, Sailik Sengupta, Daniele Bonadiman, Saab Mansour, and Arshit Gupta. 2024. FLAP: Flow-Adhering Planning with Constrained Decoding in LLMs. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 517–539. https://doi.org/10.18653/v1/2024.naacl-long.29

[81] Igor Rožanc and Marjan Mernik. 2021. The screening phase in systematic reviews: Can we speed up the process? *Advances in Computers* 123 (2021), 115–191. https://doi.org/10.1016/bs.adcom.2021.01.006

[82] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, et al. 2023. TPTU: Task planning and tool usage of large language model-based ai agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

[83] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. 2023. Identifying the risks of LM agents with an LM-emulated sandbox. *arXiv* (2023). https://doi.org/10.48550/arXiv.2309.15817

[84] Tanmana Sadhu, Ali Pesaranghader, Yanan Chen, and Dong Hoon Yi. 2024. Athena: Safe Autonomous Agents with Verbal Contrastive Learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1121–1130. https://doi.org/10.18653/v1/2024.emnlp-industry.84

[85] Avirup Saha, Lakshmi Mandal, Balaji Ganesan, Sambit Ghosh, Renuka Sindhgatta, Carlos Eberhardt, Dan Debrunner, and Sameep Mehta. 2024. Sequential API Function Calling Using GraphQL Schema. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 19452–19458. https://doi.org/10.18653/v1/2024.emnlp-main.1083

[86] Guto Leoni Santos, Vitor Gaboardi dos Santos, Colm Kearns, Gary Sinclair, Jack Black, Mark Doidge, Thomas Fletcher, Dan Kilvington, Patricia Takako Endo, Katie Liston, and Theo Lynn. 2024. Kicking prejudice: large language models for racism classification in soccer discourse on social media. In *International Conference on Advanced Information Systems Engineering*. Springer, 547–562. https://doi.org/10.1007/978-3-031-61057-8_32

[87] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.

[88] Aaditya Shah, Shridhar Mehendale, and Siddha Kanthi. 2024. Efficacy of Large Language Models for Systematic Reviews. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. 29–35. https://doi.org/10.1109/FLLM63129.2024.10852502

[89] Vasu Sharma, Karthik Padthe, Newsha Ardalani, Kushal Tirumala, Russell Howes, Hu Xu, Po-Yao Huang, Shang-Wen Li, Armen Aghajanyan, Gargi Ghosh, et al. 2024. Text quality-based pruning for efficient training of language models. *arXiv* (2024). https://doi.org/10.48550/arXiv.2405.01582

[90] Haiyang Shen, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and Yun Ma. 2024. ShortcutsBench: a large-scale real-world benchmark for API-based agents. *arXiv* (2024). https://doi.org/10.48550/arXiv.2407.00132

[91] 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. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Vol. 36. Curran Associates Inc.

[92] Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024. TaskBench: Benchmarking Large Language Models for Task Automation. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., 4540–4574.

[93] Zhuocheng Shen. 2024. LLM with tools: A survey. *arXiv* (2024). https://doi.org/10.48550/arXiv.2409.18807

[94] Ying Sheng, Sudeep Gandhe, Bhargav Kanagal, Nick Edmonds, Zachary Fisher, Sandeep Tata, and Aarush Selvan. 2024. Measuring an LLM's Proficiency at using APIs: A Query Generation Strategy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 5680–5689. https://doi.org/10.1145/3637528.3671592

[95] Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Zhumin Chen, Suzan Verberne, and Zhaochun Ren. 2024. Chain of tools: Large language model is an automatic multi-tool learner. *arXiv* (2024). https://doi.org/10.48550/arXiv.2405.16533

[96] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Vol. 36. Curran Associates Inc., 8634–8652.

[97] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. 2023. The curse of recursion: Training on generated data makes models forget. *arXiv* (2023). https://doi.org/10.48550/arXiv.2305.17493

[98] Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. RestGPT: Connecting Large Language Models with Real-World RESTful APIs. *arXiv* (2023). https://doi.org/10.48550/arXiv.2306.06624

[99] Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. 2024. Workbench: a benchmark dataset for agents in a realistic workplace setting. *arXiv preprint* (2024). https://doi.org/10.48550/arXiv.2405.00823

[100] 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* (2023). https://doi.org/10.48550/arXiv.2306.05301

[101] Chunliang Tao, Xiaojing Fan, and Yahe Yang. 2024. Harnessing LLMs for API Interactions: A Framework for Classification and Synthetic Data Generation. In *2024 5th International Conference on Computers and Artificial Intelligence Technology (CAIT)*. 628–634. https://doi.org/10.1109/CAIT64506.2024.10962957

[102] Brian Thompson and Matt Post. 2020. Paraphrase Generation as Zero-Shot Multilingual Translation: Disentangling Semantic Similarity from Lexical and Syntactic Diversity. In *Proceedings of the Fifth Conference on Machine Translation*. Association for Computational Linguistics, 561–570. https://aclanthology.org/2020.wmt-1.67/

[103] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A Controllable World of Apps and People for Benchmarking

Interactive Coding Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 16022–16076. https://doi.org/10.18653/v1/2024.acl-long.850

[104] Hongru Wang, Rui Wang, Boyang Xue, Heming Xia, Jingtao Cao, Zeming Liu, Jeff Z. Pan, and Kam-Fai Wong. 2024. 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*. Association for Computational Linguistics, 15322–15336. https://doi.org/10.18653/v1/2024.emnlp-main.856

[105] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345. https://doi.org/10.1007/s11704-024-40231-1

[106] Shufan Wang, Sébastien Jean, Sailik Sengupta, James Gung, Nikolaos Pappas, and Yi Zhang. 2023. Measuring and Mitigating Constraint Violations of In-Context Learning for Utterance-to-API Semantic Parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 7196–7207. https://doi.org/10.18653/v1/2023.findings-emnlp.478

[107] Wenxuan Wang, Juluan Shi, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, and Michael R Lyu. 2024. Learning to ask: When LLMs meet unclear instruction. *arXiv* (2024). https://doi.org/10.48550/arXiv.2409.00557

[108] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Vol. 35. Curran Associates, 24824–24837.

[109] Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *CCF International Conference on Natural Language Processing and Chinese Computing*. Springer, 372–384. https://doi.org/10.1007/978-981-97-9434-8_29

[110] Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 2024. ToolPlanner: A Tool Augmented LLM for Multi Granularity Instructions with Path Planning and Feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 18315–18339. https://doi.org/10.18653/v1/2024.emnlp-main.1018

[111] Ruixuan Xiao, Wentao Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, and Yongbin Li. 2024. FlowBench: Revisiting and Benchmarking Workflow-Guided Planning for LLM-based Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 10883–10900. https://doi.org/10.18653/v1/2024.findings-emnlp.638

[112] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. TravelPlanner: a benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org.

[113] Han Xu, Ruining Zhao, Jindong Wang, and Haipeng Chen. 2024. RESTful-Llama: Connecting User Queries to RESTful APIs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1433–1443. https://doi.org/10.18653/v1/2024.emnlp-industry.105

[114] 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* (2023). https://doi.org/10.48550/arXiv.2305.16504

[115] Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing Tool Retrieval with Iterative Feedback from Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 9609–9619. https://doi.org/10.18653/v1/2024.findings-emnlp.561

[116] Mohammad-Ali Yaghoub-Zadeh-Fard. 2021. *Scalable and Quality-Aware Training Data Acquisition for Conversational Cognitive Services*. Ph. D. Dissertation. UNSW Sydney.

[117] Mohammad-Ali Yaghoub-Zadeh-Fard, Boualem Benatallah, Fabio Casati, Moshe Chai Barukh, and Shayan Zamanirad. 2020. User utterance acquisition for training task-oriented bots: a review of challenges, techniques and opportunities. *IEEE Internet Computing* 24, 3 (2020), 30–38. https://doi.org/10.1109/MIC.2020.2978157

[118] Mohammad-Ali Yaghoub-Zadeh-Fard, Boualem Benatallah, Moshe Chai Barukh, and Shayan Zamanirad. 2019. A Study of Incorrect Paraphrases in Crowdsourced User Utterances. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 295–306. https://doi.org/10.18653/v1/N19-1026

[119] Seungbin Yang, ChaeHun Park, Taehee Kim, and Jaegul Choo. 2024. Can Tool-augmented Large Language Models be Aware of Incomplete Conditions? *arXiv* (2024). https://doi.org/10.48550/arXiv.2406.12307

[120] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. MM-REACT: Prompting chatgpt for multimodal reasoning and action. *arXiv* (2023). https://doi.org/10.48550/arXiv.2303.11381

[121] Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, Zhiyuan Liu, Xiaodong Shi, and Maosong Sun. [n. d.]. MatPlotAgent: Method and Evaluation for LLM-Based Agentic Scientific Data Visualization. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 11789–11804. https://doi.org/10.18653/v1/2024.findings-acl.701

[122] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Vol. 35. Curran Associates, 20744–20757.

[123] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv* (2024). https://doi.org/10.48550/arXiv.2406.12045

[124] 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* (2022). https://doi.org/10.48550/arXiv.2210.03629

[125] Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. 2022. MultiWOZ 2.4: A Multi-Domain Task-Oriented Dialogue Dataset with Essential Annotation Corrections to Improve State Tracking Evaluation. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, 351–360. https://doi.org/10.18653/v1/2022.sigdial-1.34

[126] Junjie Ye, Guanyu Li, SongYang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Tao Ji, Qi Zhang, Tao Gui, and Xuanjing Huang. 2025. ToolEyes: Fine-Grained Evaluation for Tool Learning Capabilities of Large Language Models in Real-world Scenarios. In *Proceedings of the 31st International Conference on Computational Linguistics*. Association for Computational Linguistics, 156–187. https://aclanthology.org/2025.coling-main.12/

[127] Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. RoTBench: A Multi-Level Benchmark for Evaluating the Robustness of Large Language Models in Tool Learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 313–333. https://doi.org/10.18653/v1/2024.emnlp-main.19

[128] Guoli Yin, Haoping Bai, Shuang Ma, Feng Nan, Yanchao Sun, Zhaoyang Xu, et al. 2025. MMAU: A Holistic Benchmark of Agent Capabilities Across Diverse Domains. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, 4737–4765. https://doi.org/10.18653/v1/2025.findings-naacl.267

[129] Shayan Zamanirad, Boualem Benatallah, Carlos Rodriguez, Mohammadali Yaghoubzadehfard, Sara Bouguelia, and Hayet Brabra. 2020. State machine based human-bot conversation model and services. In *Advanced Information Systems Engineering (CAiSE)*. Springer, 199–214. https://doi.org/10.1007/978-3-030-49435-3_13

[130] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 10471–10506. https://doi.org/10.18653/v1/2024.findings-acl.624

[131] Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. 2024. Ask-before-Plan: Proactive Language Agents for Real-World Planning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 10836–10863. https://doi.org/10.18653/v1/2024.findings-emnlp.636

[132] Yinger Zhang, Hui Cai, Xierui Song, Yicheng Chen, Rui Sun, and Jing Zheng. 2024. Reverse Chain: A Generic-Rule for LLMs to Master Multi-API Planning. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 302–325. https://doi.org/10.18653/v1/2024.findings-naacl.22

[133] Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. [n. d.]. ToolBeHonest: A Multi-level Hallucination Diagnostic Benchmark for Tool-Augmented Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11388–11422. https://doi.org/10.18653/v1/2024.emnlp-main.637

[134] Yazhou Zhang, Mengyao Wang, Qiuchi Li, Prayag Tiwari, and Jing Qin. 2025. Pushing The Limit of LLM Capacity for Text Classification. In *Companion Proceedings of the ACM on Web Conference 2025*. Association for Computing Machinery, 1524–1528. https://doi.org/10.1145/3701716.3715528

[135] Zekai Zhang, Yiduo Guo, Yaobo Liang, Dongyan Zhao, and Nan Duan. 2024. PPTC-R benchmark: Towards Evaluating the Robustness of Large Language Models for PowerPoint Task Completion. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 12387–12402. https://doi.org/10.18653/v1/2024.findings-emnlp.722

[136] Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. ToolRerank: Adaptive and Hierarchy-Aware Reranking for Tool Retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. ELRA and ICCL, 16263–16273. https://aclanthology.org/2024.lrec-main.1413/

[137] Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, et al. 2023. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv* (2023). https://doi.org/10.48550/arXiv.2308.07921

[138] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv* (2024). https://doi.org/10.48550/arXiv.2406.15877