Data Science and Engineering
https://doi.org/10.1007/s41019-025-00296-9

REVIEW/SURVEY PAPERS

LLM-Based Agents for Tool Learning: A Survey

Weikai Xu[1] · Chengrui Huang[1] · Shen Gao[1] · Shuo Shang[1]

Received: 9 October 2024 / Revised: 5 April 2025 / Accepted: 21 April 2025

Abstract
Human beings capable of making and using tools can accomplish tasks far beyond their innate abilities, and this paradigm of integration with tools may not be limited to humans themselves. Recently, the large language model (LLM) has demonstrated immense potential across various fields with its unique planning and reasoning abilities. However, there are still many challenges beyond its capabilities due to deficiencies in its training data and inherent illusions. Thus, integrating LLMs and tools into tool learning agents has become a new emerging research direction. To this end, we present a systematic investigation and comprehensive review of tool-learning agents in this paper. We start by introducing the definition of the tool learning task for Agents and then illustrating the typical architecture of the tool-learning models. Since these tools are all defined by users, LLM does not know what tools there are and what their functions are. Thus, LLMs should first find appropriate tools and split the tool retrieval methods into two categories: training-based and non-training-based. To accurately complete the user task, it is important to decompose the task into several sub-tasks and execute them in the correct order. Following that, we introduce the tool planning methods and organize these works by whether they rely on the model's inherent reasoning capabilities for planning or utilize external reasoning tools. Due to the rapid development of this field, we also introduce an emerging frontier direction: using multimodal tools for LLM. In addition, we compile current open-source benchmarks and evaluation metrics, focusing on their scale, composition, calculation methods, and assessment dimensions. Next, we introduce several application scenarios for the LLM-based tool learning methods. Finally, we discuss the safety and ethical issues involved in tool learning.

Keywords Large language model · Tool learning · Agent · Vision-language model

1 Introduction

Tool- learning is considered a unique ability of humans [4]. The innovation and application of tools expand the range of available resources, empowering humans to overcome physiological limitations and enhancing problem-solving efficiency [37]. Recently, LLMs have demonstrated impressive abilities in reasoning and comprehension. There is a growing interest in their potential to interact with the real world and enhance their functionality through integration with external tools. Inspired by human tool learning, the integration of tool-learning capabilities into LLMs has received extensive attention in the literature. The goal of tool learning is to empower LLMs to understand the functionality of various tools and employ them effectively to accomplish specific tasks.

The integration of multiple inputs, multi-agent planning strategies and reasoning frameworks, along with diverse tool repositories, facilitates the development of tool learning capabilities within LLMs. Thus, equipping LLMs with tools has become a new emerging research direction. Currently, research primarily focuses on two approaches for integrating LLMs with tools. The first approach focuses on empowering LLMs to master the use and application of tools, thereby enhancing LLMs intelligence. The second approach endeavors to view tools as extensions of LLMs inherent capabilities. The two approaches share the same three steps to augment LLMs with tools: invoking tools at the right time, retrieving the appropriate tools, and effectively utilizing them to solve real-world tasks. However, each of the three steps in tool-learning presents several challenges: (1) LLMs often struggle to identify appropriate tools and utilize them effectively; (2) agents based on LLMs still face challenges

✉ Shuo Shang
jedi.shang@gmail.com

[1] University of Electronic Science and Technology of China, Chengdu, China

Published online: 26 June 2025

in invoking tools in multi-step reasoning problem scenarios and planning tools appropriate; (3) equipping LLMs with tool-use capabilities through text input can lead to ambiguity in understanding users' true intentions.

In this paper, to bring further insights into the development of tool learning, we undertake a detailed investigation and thorough analysis of tool learning, seeking to develop an in-depth understanding of the key challenges, the progress of current work, and future directions in this area. Before delving into existing methodologies, we first introduce the definitions of tool learning within LLMs (Sect. 2.1), which encompass three main components: timing for tool invocation, retrieving tools, and tool usage methods. Following that, faced with multi-inputs, diverse tools, and complex real-world tasks, we outline a basic structured framework (Sect. 2.2) to enhance LLMs' proficiency in tool utilization. In this section, we provide a detailed introduction to constructing a tool-learning environment based on user queries. After receiving feedback signals from the environment, the structured framework filters input data, employs reflection to address hallucination, and ultimately plans and utilizes tools. With this foundational knowledge in place, we review current works according to the three steps of tool learning. Firstly, LLMs need to retrieve target tools based on user commands. However, the diversity of tool repositories and LLMs' limited understanding of tool functionality make accurately retrieving specific tools challenging. Thus, we categorize existing works based on whether they enhance the model's ability to retrieve tools through training. For open-source LLMs, LLMs attain generalized tool-use abilities through demonstrations and tool-use corpus without training. For closed-source LLMs, supervised training is utilized to attribute LLMs' tool-learning abilities. Secondly, LLMs often break down real-world complex tasks into sub-tasks and solve each individually. When integrating LLMs with tools, the lack of tool planning, such as selecting tools for sub-tasks and ordering tools, makes it difficult to invoke tools effectively in multitasking scenarios. Thus, we organize the process based on whether it relies on the model's inherent reasoning capabilities for planning or utilizes external reasoning tools (Sect. 4). Thirdly, text-based tool learning cannot clearly understand the user's intentions, limiting LLMs' ability to address multimodal problems. Thus, we discuss how LLMs interact with the environment and feedback using multimodal tools (Sect. 5). Beyond these three steps of tool learning, tools enable LLMs to understand multi-modal inputs and generate code, thereby facilitating their application in various downstream tasks (Sect. 6), including data science [71, 167], software engineering [3, 92], web and mobile development [123, 155], and robot control [81, 129]. Nam et al. [92] propose integrating LLMs into IDEs, assisting with code generation tasks such as explaining code and providing details of APIs, among other functionalities. Zhou et al. [167] introduce using vector databases to reduce the cost of employing LLMs (Fig. 1).

To present a comprehensive overview of the resources and developments within tool learning, we separately compile the **datasets**(Sect. 7.1) and **evaluation metrics**(Sect. 7.2) specific to this field. Finally, we discussed the challenges of applying these advancements to the real world scenarios(§8). Considering the application constraints in the real world, we introduced **safety problems** and **ethical responsibilities** involved in tool learning agents. Based on the various challenges that tool learning currently faces and the development status of LLMs, we present potential **future directions and**

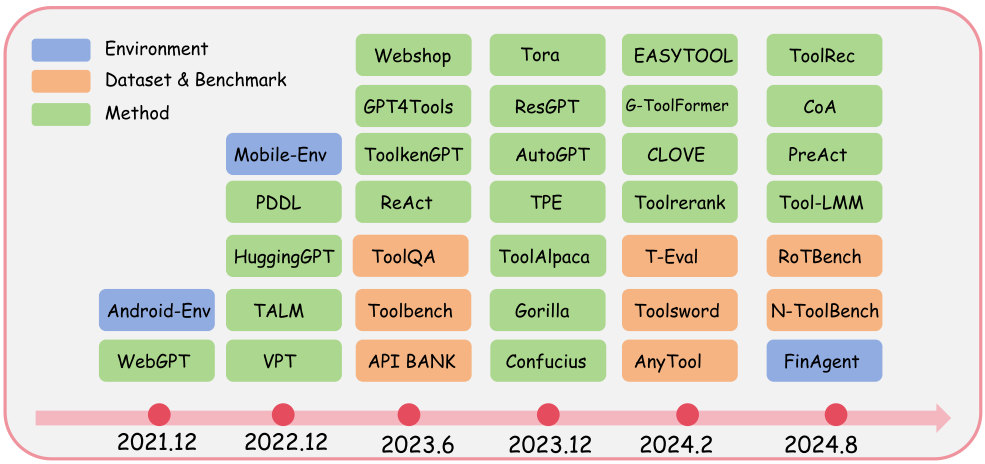

Fig. 1 An illustration of the development trajectory of tool learning. We use blue, orange, and green to represent engineering-focused papers, dataset and evaluation papers, and methodological papers, respectively. For each period, we have selected a range of representative landmark studies that have significantly contributed to the field. (Note that the time points are not perfectly aligned and may have slight deviations)

application scenarios for tool learning at the end of the survey.

2 Preliminary Knowledge

In this section, we primarily introduce preliminary knowledge about tool learning, including a representative definition of an end-to-end tool-used task(Sect. 2.1) and a current, widely-used framework for these tasks(Sect. 2.2).

2.1 Task Definition

In this section, we will explore what constitutes the foundational tasks of tool learning. Before designing a solution, we need to know what task needs to be accomplished. The basic tool learning tasks can be encapsulated by the "*Three Ws*": **W**hether, **W**hich, and Ho**W**. As shown in Fig. 2, the essential judgments to be made involve determining whether the tool call is necessary, which tools to retrieve, and how to utilize these tools effectively.

Tool Invocation The decision on whether to use a tool when addressing a task was initially defined by Schick [108], who posits that the simplest standard for tool usage is whether the model can correctly answer the question directly without tools assistance. However, this is often challenging because the LLM itself may not be able to recognize its errors. Although during the training stage, the model can be optimized using Batch Reinforcement Learning by relying on multi-stage process feedback and offline annotations derived from the final answer, in more practical scenarios, especially in real-time online settings, such feedback is difficult to obtain. As a result, most benchmarks involving tools do not consider the necessity of tool invocation separately. Instead, they assume that the entire problem environment requires tool assistance for resolution. Subsequently, Du [29] further expanded the definition of this task. He argues that the initial step should include evaluating whether the current toolsets contain tools that can facilitate problem-solving, as it is necessary to consider the cost of tool invocation and the efficiency of task resolution. In simple tasks, the error rates of tool retrieval and parameter input are often higher than the task. In contrast, complex problems typically involve the retrieval and invocation of multiple tools, which often results in high reasoning costs.

Tool Retrieval Tool retrieval is defined by Qin et al. [103] as the process by which LLMs decompose the task into multiple subtasks based on queries, generate solutions for each subtask, and incrementally schedule appropriate tools to produce the answers. Each subtask plan includes a structured retrieval process within the toolset. However, Gao and Gou [33, 39] argue that tool retrieval should involve a process of comparative screening cross-categories, in-categories, and among individual tool responses rather than directly retrieving a single target. This process is necessary because of the difficulty in keeping data labeling in sync with toolset updates. When retrieving tools outside the distribution of the original toolset, the LLMs often struggle to get the correct tool directly. A more common method is conducting cross-category searching, followed by in-category comparison based on their responses. This method is particularly common in mathematical and coding problems, where the differences between tools within the same category are minimal, making the order of tool invocation more important.

Tool Usage LLMs rely on tool documentation to understand how to invoke tools and interpret their return formats, making the content and organization of the tool documentation crucial. Tool documentation generally includes two types of information: (1) Background knowledge of a tool, which provides a rough description of the tool's functionality and parameter definitions; (2) Demonstration examples of a tool, which include *query-tool invocation-response* triplets. The former information helps the model compare the complexity between similar tools, while the latter aids the model

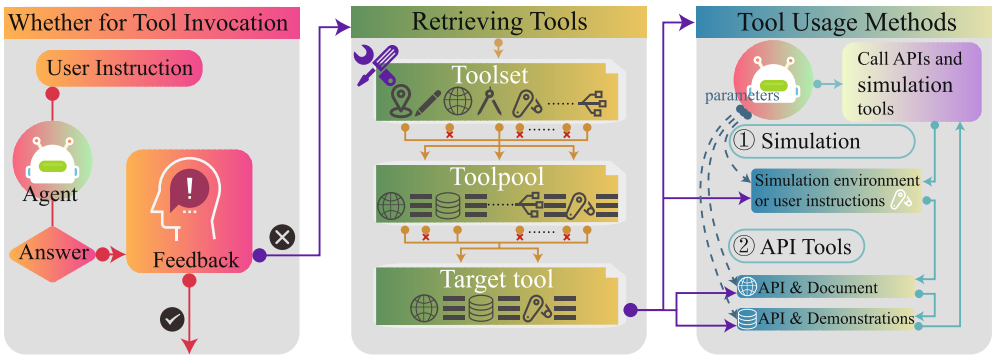

Fig. 2 The three components of the foundational task include: determining **whether** to use a tool, selecting **which** tool to use, and understanding **how** to effectively employ these tools

in understanding the input–output relationships of a specific tool. In fact, tool documentation gradually becomes outdated as tool invocation methods and data sources change. Therefore, most toolsets provide additional methods for each tool to verify parameter accuracy and return statuses.

To further illustrate the entire process, Table 1 demonstrates how three tasks are integrated throughout the problem-solving workflow when the user asks: *"Can you find me a flight from New York to San Francisco and let me know if I need to pack an umbrella."*

In contrast to the example in the table, many tasks involve dependencies between multiple tools, where the order of their invocation is crucial for problem-solving. For instance, before querying the elevation of a specific location, you first need to know its latitude and longitude. Additionally, the tool selected by the model might fail to provide the expected answer because the invocation method may no longer be valid, or the information returned by the tool may no longer match the original documentation. In the next subsection, we will discuss how to avoid these issues.

2.2 Overall Framework

The process described in the example above represents the ideal scenario. A multi-step framework is needed to ensure that each step is as accurate as possible. Through this standard framework, LLMs can enhance their proficiency in tool-used tasks. To this end, we first introduce the whole tool learning agent framework and then explain how each component contributes to it.

As shown in Fig. 3, the entire framework consists of five components: the executor, perceiver, validator, controller, and retriever. Their targets are the environment, invocation results, feedback, the agent, and the toolset. The perceiver and validator are used to assess the current state of the environment, and the controller adjusts the plan and target tools based on their reflection and perception. Subsequently, the retriever gradually narrows down the pool of candidate tools from the toolset, ultimately determining the target tool based on the documentation and invocation feedback. As the circular diagram illustrates, these five components iteratively take turns until all subtasks are resolved.

Controller&Agent The primary functions of the controller are as follows: 1) decomposing the user's input into multi-step subtasks, 2) formulating solutions for each subtask, and 3) adjusting the plan based on feedback and checking if the task has been completed. The controller is the head of the tool learning agent. The comprehension of user intent is once primarily facilitated by an intent distribution model [18], which decomposes the user's commands into tasks for specialized sub-models to execute. When LLM obtains the

Table 1 This example involves multiple tools, encompassing tool invocation determination, planning, retrieval, and necessity of usage

Step	Action	Details
1. User Input	User requests flight information and weather conditions. LLM determines whether tool invocation is needed to answer this question	"Can you find me a flight from New York to San Francisco and let me know if I need to pack an umbrella"
2. Overall Plan	LLM makes the overall plan based on the user input	The user wants to know weather and flight information, but there is no obvious order between these two needs. Just call the corresponding tools in order
3. Flight Search Tool Retrieval	The LLM retrieves the most relevant tool from the tool library, named the "flight search API"	`API Call: {tool: "flight_search_api", action: "search_flights", parameters: {origin: "New York", destination: "San Francisco", date: "2024-09-01"}}`
4. Flight Information Result	The tool returns available flight options	1. Flight: AA100, 08:00 AM, $250 2. Flight: UA200, 09:00 AM, $230
5. Weather Tool Retrieval	The LLM retrieves the most relevant tool from the tool library, named the "Weather API."	`API Call: {tool: "weather_api", action: "get_weather", parameters: {city: "San Francisco", date: "2024-09-01"}}`
6. Weather Information Result	The tool returns the weather forecast	Rainy, High: 20 C, Low: 15 C
7. Final Output	System provides a comprehensive response to the user	"I found two flights from New York to San Francisco on September 1st. You can take American Airlines flight AA100 departing at 08:00 AM for $250, or United Airlines flight UA200 departing at 09:00 AM for $230. The weather in San Francisco is expected to be rainy, so it's a good idea to pack an umbrella"

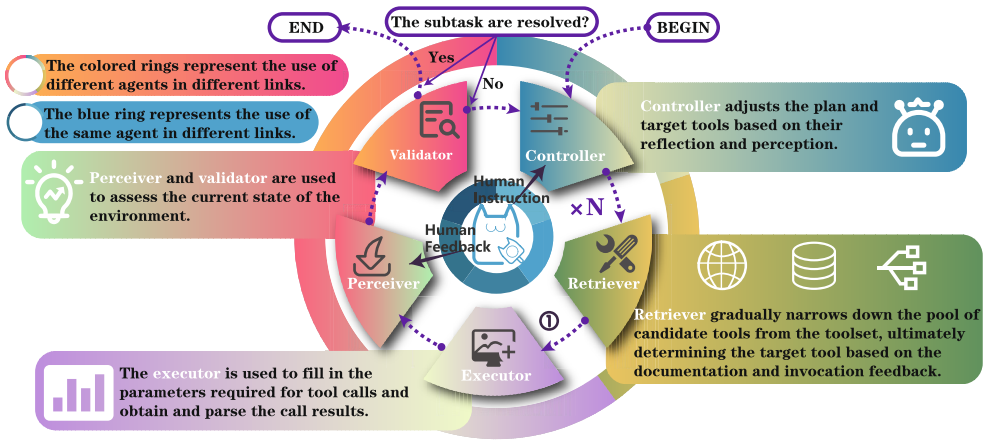

Fig. 3 Illustration of the tool-learning agent framework, where we display five core components: toolset, Controller Agent, Perceive, Reflection and Environment

user's intent through semantic analysis, it can opt for structured searching by category or rerank the tools based on their functionality [165]. Subsequently, it plans the sequence in which these tools should be used through reasoning. The function of the controller can be described by the equation 1:

$$(Plan_{r+1}, T_{r+1}) \leftarrow Controller(Plan_r, H_t, \mathrm{f}_r, \mathrm{R}_r, \mathcal{Q}) \tag{1}$$

where $Plan_{r+1}$ and T_{r+1} denote the plan and tool pool selected for the next iteration, $Plan_r$ represents the current plan, H_t is the history of tool invocations, f_r is the response from the perceiver, and R_r captures the validator's reflection process.

Retriever&ToolSet The toolset $\mathcal{T}$ is defined as a collection of tools, which can be represented as $\mathcal{T} = \{t_1, t_2, \ldots\}$, where each tool t_i is characterized by a tuple $t_i = \{i, c, p, f\}$. In this tuple, i denotes the invocation context, c represents the category of the tool, p indicates the parameters required for operation, and f describes the functionality of the tool. In most tool learning scenarios, the tool t_i typically denotes an API Call, for instance, a weather query API. Within this context, i, c, p, and f each correspond to distinct aspects of the tool's functionality. Specifically, i details the mode and source of weather forecast information, c categorizes the API into its broad and specific types, p outlines the parameters required for the call, such as region and time, and f provides a comprehensive explanation of the functionality. In a rarer scenario, the tools employed can directly interact with the real world, such as robotic arms, which require more complex torque parameters and detailed user manuals for effective operation. Only a few tools are available in this toolset, with the extreme complexity of individual tools and the involvement of sensor physical outputs and motor-simulated signal inputs. In summary, the Retriever needs to find the target tool within the toolset according to a specified rule.

Executor&Environment We define the Environment $\mathcal{E}$ as the tube for user query inputs and the operational domain for tools. This environment may manifest as a digital world, virtualized in nature, or a tangible world, subject to physical laws. For example, a user's statement represents the operational environment for a weather tool, such as "*I want to know the weather in Haidian, Beijing, next Friday.*". In contrast, the environment could be a simulation software or a real-world physical scenario for a robotic arm. We illustrate the role of the environment with Eq. 2:

$$f_E \leftarrow t_i(p_i \mid \mathcal{E}, Plan, \mathcal{Q}, H_t) \tag{2}$$

where H_t denotes the function of tool invocations, $Plan$ refers to the current plan of tools using, and $\mathcal{Q}$ represents the user's query. This equation represents the feedback f_E provided to the subsequent component by the environment $\mathcal{E}$ after operating the tool T_i with parameters p_i. The Executor is responsible for executing the tools in the corresponding environment and parsing the returned content according to the specified format.

Perceiver&Feedback We define the perceiver as a component designed to receive feedback from the environment, transforming this information into a standardized format for downstream utilization. In the early stages, this feedback, including penalties or rewards, was limited to processing textual information. However, in more complex tasks, sensors equipped with various encoding capabilities can accept

a quartet of multimodal signals [168], including text(1 d), audio, video(3 d), and images(2 d).

Validator&Reflection The concept of reflection was inspired by the observation that individuals often find it challenging to identify their own mistakes while readily spotting others' errors [96, 125]. Reflection can be divided into unsupervised self-reflection and supervised external reflection. In the single-agent framework, unsupervised self-reflection ensures parameters are correctly filled and results are accurately returned, allowing for the timely deletion or updates of incorrect tools. For instance, in response to a user query such as "*Help me check the weather in Beijing next Friday*," LLM needs to assess the utility of interpreting "*next Friday*" as the specific date "*March 2nd*" in facilitating problem resolution. In the multi-agent framework, a more powerful agent is used as a supervised external reflection to evaluate the appropriateness of tool invocations, using its assessments as gold answers. In a mobile scenario, after the controller invokes an API on the phone, a dedicated validator can retrieve the click API invocation results from the system logs.

Consider a cyclic process where the controller repeatedly executes the above steps until a specific termination condition is met. However, in some works, this step is also handled by the Validator.

Algorithm 1 The standard framework of Tool learning

Require: User input query $\mathcal{Q}$
Ensure: Final answer Ans, Tool using history H_t
1: Initialize environment $\mathcal{E}$, history H_t,toolset $\mathcal{TS}$
2: Initial plan $Plan_0$ and set state $r \leftarrow 0$
3: **while** do not get the final answer **do**
4: Parse user input and current state to update plan and select tool Pool:

$$(Plan_{r+1}, T_{r+1}) \leftarrow Controller(Plan_r, H_t, f_r, R_r, \mathcal{Q}, \mathcal{TS})$$

5: Execute tool T_{r+1} and obtain Ans A_{r+1}:

$$A_{r+1} \leftarrow t_{r+1}(p_{r+1} \mid \mathcal{E}, Plan_{r+1}, \mathcal{Q}, H_t)$$

6: Update history to H_{t+1}, state to $r \leftarrow r+1$ and answer $Ans \leftarrow A_{r+1}$
7: Calculate the cumulative loss from discrepancies between the outcomes of tool invocations $A_{r+1,i}$ and the expected results from either human judgment or an optimal language model $f_{\text{human},i}$ / $f_{\text{olm},i}$, utilizing the log loss function:

$$f_{r+1} \leftarrow -\sum_{i=1}^{N} [A_{\text{human},i} \log(A_{r+1,i}) + (1 - A_{\text{human},i}) \log(1 - A_{r+1,i})]$$

8: Update reflection R_{r+1} based on the feedback f_{r+1}
9: **end while**

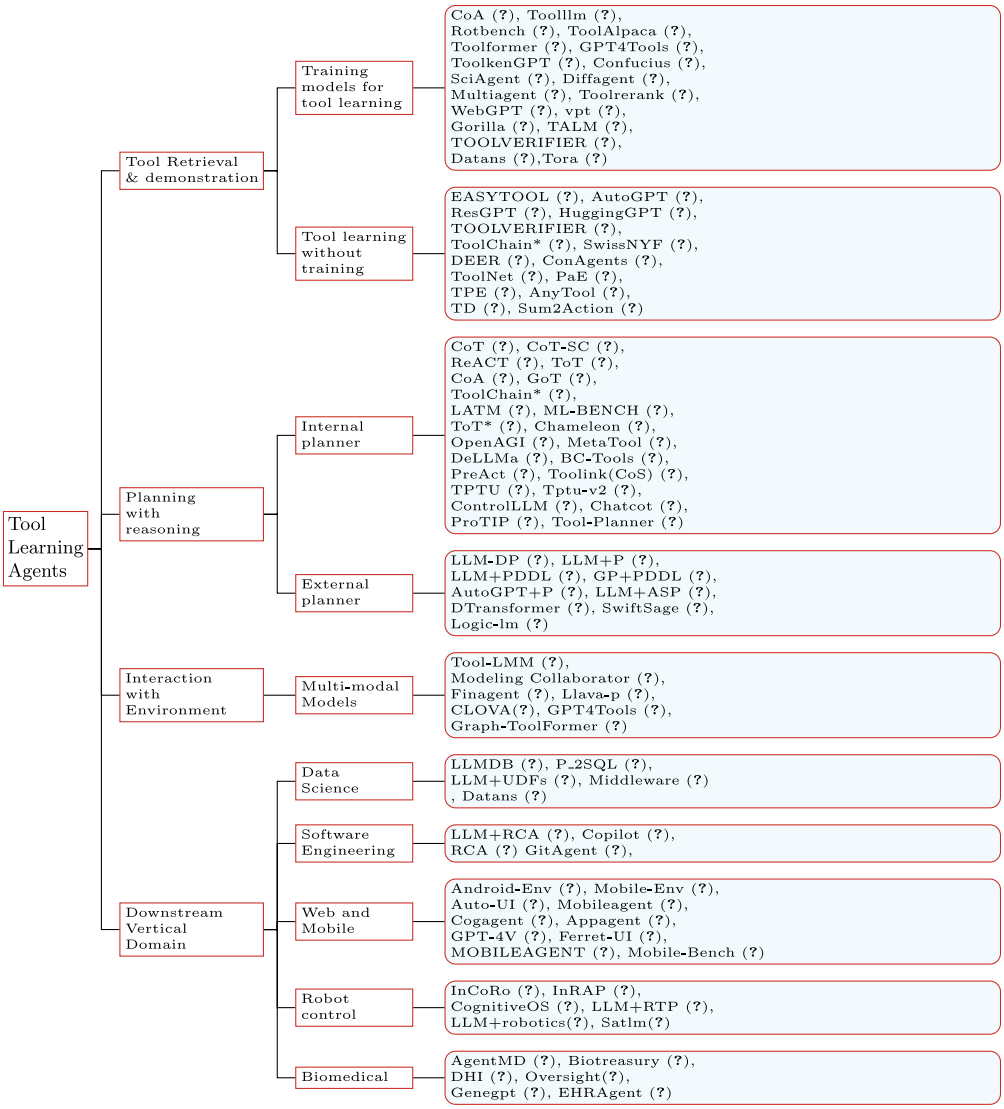

Fig. 4 Tool learning work summary tree diagram from 2022.6 to 2024.5. Since some papers have not named their works, we have provided abbreviated names for ease of reference

Whole framework The entire framework of the tool-learning agent is outlined in Algorithm 1, with the reflection component being a unique feature of the multi-agent framework. With multimodal feedback from the environment, the reward mechanism becomes significantly more complex; thus, the algorithm presented serves as just a general representation. According to the order of presentation in subsequent chapters, we list the related tool-learning works in Fig. 4. Some works involve innovations multiplied, so they may appear in multiple categories.[1]

3 Learning to Use Tools

Existing tool learning agents either primarily rely on closed-source LLMs, such as GPT-4 [1], to attain generalized tool-use abilities without training or utilize supervised learning to train limited scopes of tools on open-source models [102]. For the former, it is crucial to construct appropriate tool demonstration and a diverse tool-use corpus; for the latter, the core problems are generating effective supervised data and defining what constitutes an efficient fine-tuning task. All the work in this section is presented in Fig. 5.

[1] Some works may appear in multiple categories because they may have multiple innovations. For example, GPT4 Tools involves both downstream task fine-tuning and multi-modal input.

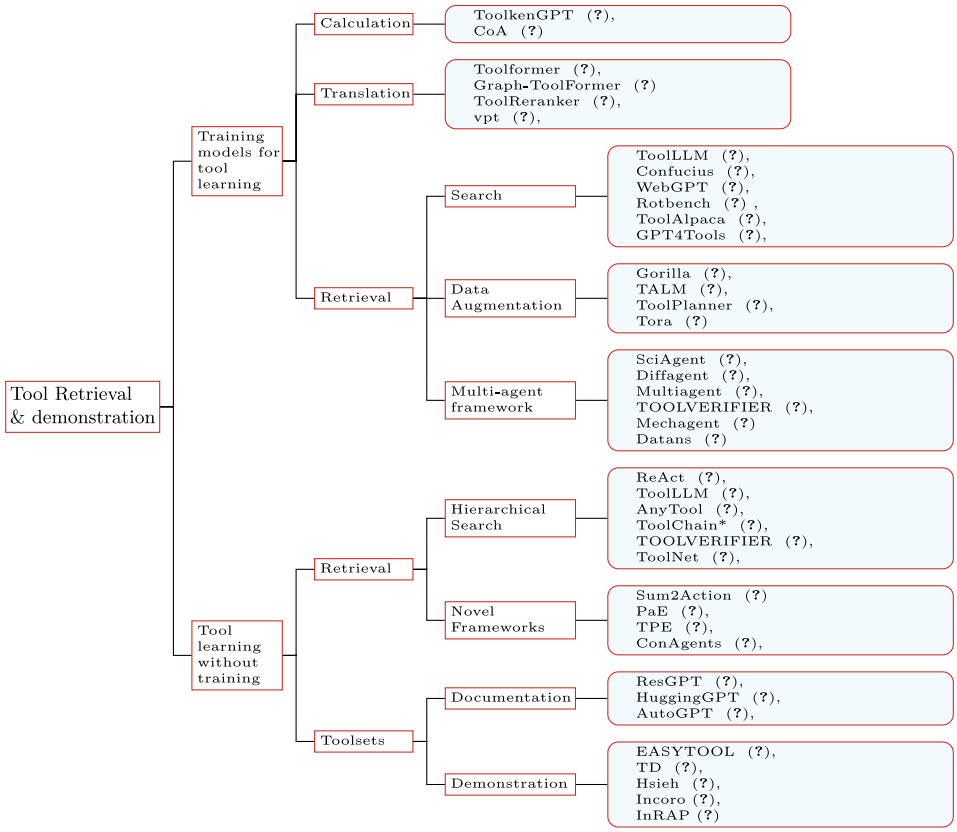

Fig. 5 The detailed categorization of all the work in the "Learning to use tools" section is provided. However, due to space limitations, not all work shown in the figure will be discussed in detail in the subsections

3.1 Training Models for Tool Learning

To enhance a model's capability to utilize tools, some methods fine-tune the model on datasets designed explicitly for tool learning. These can be split into three categories: **retrieval, calculation, and translation**. Few works are pre-training on LLM, such as LLaVA-Plus, because of expensive training costs and hard-to-define pre-training tasks. Other works, such as Tora [39], use teacher LLMs to perform knowledge distillation on tool learning agents for shaping their output space. Below, we will discuss the above works in detail.

3.1.1 Calculation

ToolkenGPT [46] is the first work to predict the next tool by calculating the probability of tool special tokens. Specifically, each tool with its "toolken embeddings" is trained based on a large number of contextual examples and basic descriptions, adjusting it to align with normal text tokens. After getting the trained toolken $W_\tau \in \mathbb{R}^{|\mathcal{T}|\times d}$, the LLM is in the reasoning mode by default, generating the next token, which considers word tokens and toolkens uniformly. The LLM predicts the next token with the probability as follows:

$$P(t_i|t_{<i}) = \text{Softmax}([W_v;W_\tau] \cdot h_{i-1}) \tag{3}$$

where the next token can be either a word token or a toolken, i.e. $t_i \in \mathcal{V} \cup \mathcal{T}$, and [;] is the concatenation operation. While the next predicted token is toolken, LLM will replace the current toolken with the corresponding tool invocation result. However, ToolkenGPT lacks an overall and effective tool usage plan to deal with interconnected tool calls. CoA [34] trains LLMs to first decode reasoning chains with abstract placeholders, and then call domain tools to reify each reasoning chain by filling in specific knowledge. This method avoids the hallucinations of purely general token prediction and preserves the LLM's reasoning capabilities. There are two main issues with unifying tool prediction as token prediction: (1) classification-based retrieval can effectively avoid cross-category tool invocation errors, but this is difficult to maintain in calculation methods. (2) the number of toolkens is correlated with the tools, suggesting that the model's general text capabilities will be affected in large-scale toolsets.

3.1.2 Translation

Translation refers to converting the tools invoked into a fixed format and embedding them into the text. Toolformer [109] is the first work with the translation free of the fine-tuning

dataset. This is because it can convert any dataset $\mathcal{C}$ into a dataset $\mathcal{C}*$ augmented with API calls. Specifically, it randomly calls the API through sampling and calculates the loss based on the call feedback. Better results are translated and recorded into text for LLM learning. This ensures that the pre-training and fine-tuning data remain strictly consistent in format. Given an API call c with a corresponding result r, API augmented $\mathcal{C}*$ can be represented as:

$$\begin{aligned} \mathrm{e}(c) &= <\texttt{API}> a_c(i_c) <\texttt{/API}> \\ \mathrm{e}(c,r) &= <\texttt{API}> a_c(i_c) \rightarrow r <\texttt{/API}> \end{aligned} \tag{4}$$

where "< API >", "< /API >" and "→" are special tokens during training stage. The challenge with this method lies in Toolformer's effectiveness, which highly depends on the availability of external tools and the reliability of training data transformation. Each tool must be accurately described and updated in real time during the transformation process. Meanwhile, original LLMs may become ineffective due to tool updates and domain shifts.

3.1.3 Retrieval

Tool retrieval capabilities refer to LLMs' structural search accuracy in the toolset. Methods to enhance a model's retrieval capabilities primarily include greedy search training, staged search training, re-ranking and truncation, and data augmentation training. The following equations can represent the entire retrieval process:

$$t^* = \arg\max_{t \in \mathcal{P}} \mathrm{Score}(t), \quad \mathcal{P} = \mathrm{Select}_{\mathrm{Pool}}(\mathcal{T}, \mathrm{R}) \tag{5}$$

while $\mathcal{P}$ is the tool pool, Score is the scoring function for evaluating the invocation results, and t^* is the target tool.

ToolLLM [103] is the first work to perform greedy search DFSDT (Depth First Search-based Decision Tree) on RapidAPI, while the visited state is annotated by the then-state-of-art GPT-4. It fine-tunes the model using coarse-grained and fine-grained classification losses and API classification loss. However, since the node-value evaluations in its training data are entirely performed by GPT-4, their accuracy is difficult to guarantee, impacting the overall quality of the training data. Experimental results show that only a very small number of tasks received ideal responses through tool invocation, while the remaining tasks failed to complete retrieval and invocation requests within the presetting number of steps. To further improve retrieval accuracy, Confucius [33] embodies the learning process of toolLLM as a multi-stage process from basic to advanced and provides corresponding training data construction methods. Specifically, these multiple training stages can be divided into warm-up training, in-category training, and cross-category training, which use different synthetic data on different stages. This staged process, from easy to difficult, gradually builds the agent's ability to distinguish between tools, as it requires progressively understanding the subtle differences among tools within the same category and distinguishing between categories at varying granularities. They also propose the Iterative Self-instruct from Introspective Feedback (ISIF) to dynamically construct training data, which updates the training dataset continuously based on model knowledge of tools. For accelerating the searching process, ToolReranker [165] uses a dual-encoder retriever to obtain coarse-grained retrieval results $C = [c_1, \ldots, c_m]$; then it applies Adaptive Truncation to C to obtain the truncated results T; finally, it uses a cross-encoder reranker to rerank T and obtains the reranked results R. ToolReranker significantly outperforms ToolLLM in search efficiency because it requires almost no multi-round feedback evaluation.

Data augmentation training for tool learning can primarily be categorized into two approaches: (1) input augmentation and (2) output space correction. Input augmentation enhances the model's retrieval and reasoning abilities by enriching the input data. For example, Gorilla [98] attaches the retrieved API documentation directly to user command data, effectively providing contextual reference information to improve the retrieval accuracy of the LLM. Specifically, the prompt given to the model explicitly instructs it to reference provided API documentation: "Use this API documentation for reference: retrieved API doc". Additionally, to boost the model's capability in filling API parameters accurately, Gorilla decomposes complex parameter-filling tasks into multiple conversational turns, each focusing on specific subsets of parameters. By fine-tuning the LLM on these instruction-driven dialogues, the model learns to incrementally reason about and correctly fill in API parameters, thus enhancing robustness and accuracy in practical scenarios. TALM [95], on the other hand, proposes an iterative self-play method for data augmentation. Starting from a limited set of labeled examples, TALM repeatedly performs a self-play procedure where the tool-augmented model generates intermediate training examples, progressively bootstrapping itself to higher-quality datasets. This iterative approach enables TALM to significantly expand and diversify the training set without extensive manual labeling, thereby effectively addressing the data scarcity problem commonly encountered in tool learning scenarios.

Output space correction methods aim to enhance model performance by identifying and rectifying erroneous predictions in model outputs, thus directly improving the quality of the training labels. For instance, Tora [39] systematically identifies errors in the diverse output space of generated tool invocation results. After error detection, these incorrect examples undergo correction-either through manual annotation or automatic validation-and subsequently get integrated back into the training dataset. This approach effectively

mitigates error propagation during training, thereby significantly improving the final model's tool-invocation accuracy and stability.

The multi-agent framework has brought new possibilities and dimensions to tool learning by enabling the decomposition of complex tasks into specialized subtasks handled by dedicated agents. For instance, Shen et al. [111] propose decomposing the general multi-agent framework into three distinct roles: a planner, a caller, and a summarizer. Each role is instantiated by a single LLM agent, with each agent specializing in its designated task and collaboratively interacting with other agents to fulfill the overarching goal. To optimize the performance of these agents, they adopt a two-stage training strategy: initially, a backbone LLM undergoes pretraining on the entire dataset without differentiating between subtasks, ensuring foundational task understanding; subsequently, the model undergoes continual fine-tuning tailored specifically to each subtask, enhancing specialized performance and inter-agent coordination. Similarly, Mechagents [93] employs a simplified yet effective two-agent framework consisting of a code-writer and an executor/self-corrector. This design enables the agents to iteratively generate, execute, and self-correct code, particularly focusing on applying finite element methods to classical elasticity problems. Through their collaborative iterations, the agents significantly improve computational reliability and accuracy, demonstrating the power of agent specialization and iterative self-correction mechanisms.

Beyond general-purpose frameworks, recent efforts have also emerged in fine-tuning models for specialized fields. For example, Sciagent[84] fine-tunes its underlying AI models using scientific reasoning datasets, significantly enhancing its capabilities in complex scientific problem-solving and reasoning tasks. Similarly, Datans[120] fine-tunes models using domain-specific financial data, thus improving the agent's proficiency in leveraging mathematical and analytical tools to analyze and interpret intricate financial datasets. Such domain-specialized fine-tuning underscores the value of targeted datasets in elevating the performance of multi-agent systems within specific vertical applications.

3.2 Tool Learning Without Training

To enhance the tool learning capabilities of models without training, there are primarily two approaches: a) optimizing the overall framework of the tool agent as mentioned in Sect. 2.2; b) retrieving and calling external tools by providing documentation and few-shot demonstrations of their functionality. For the former, these methods focus on the optimization points, including how to make a tool usage plan (Sect. 4), how to interact with the environment (Sect. 5), and how to search target tools (Sect. 3.2.1). For the latter, these methods struggle with limited context length and face difficulties when handling unusual tools (Sect. 3.2.2).

3.2.1 Tool Retrieval Methods

The optimization of closed source models in tool retrieval mainly focuses on optimizing the structured search algorithm and the sampling strategy of tool usage examples. Some works also focus on tool feedback verification and new tool generalization.

Hierarchical Search The earliest works on tool retrieval do not consider the structure of the toolset itself. ReAct [146] organizes the framework of using tools into two parts: reasoning and action. However, its reasoning module is not externalized as a search structure; instead, it leverages the LLM's inherent reasoning capabilities to specify the next tool directly. ToolLLM [103] discovered RapidAPI's special tree structure. Specifically, its organization is divided into three distinct tiers: the first tier is the domain level, encompassing various domains such as "*sports*" and "*finance*"; the second tier, designated as the category level, consists of tools that belong to specific categories; and the third tier focuses on individual APIs, with each API belonging to a specific tool. ToolLLM utilizes this characteristic to construct a Depth First Search-based Decision Tree (DFSDT). It uses DFS for retrieval and employs GPT-4 to determine the status of visited nodes, performing pruning when necessary. The core drawback of this method is its instability. While DFS can reduce search costs when the tool invocation path is superior to subpaths, it loses the ability to prioritize peer nodes in other scenarios (as in BFS and MCTS). Moreover, since the LLM does not know whether the low scores of the current node are due to a retrieval error or the absence of an appropriate tool altogether, the results of the status evaluations are not sufficiently objective. To fix this, AnyTool [29] configures LLM agents for each category of toolsets at the second tier(this tier is the most prone to errors.), providing a corresponding introduction document and demonstration. Additionally, it records the tools that failed during the retrieval process and the corresponding tasks into the corresponding document through self-reflection.

The following works further optimize the evaluation method of node states during the search process based on ToolLLM. ToolChain* [170] presents an efficient tree search-based planning algorithm tailored for large language model (LLM) agents interacting with diverse tools, such as functional APIs. It conceptualizes the entire action space as a decision tree, where each node signifies a potential API function call within a solution plan. To navigate this expansive action space effectively, ToolChain* integrates the A* search algorithm with a task-specific cost function. This design enables the model to prune high-cost branches that may involve incorrect actions, thereby identifying the most

cost-effective valid path as the solution. Extensive experiments across multiple tool-use and reasoning tasks have demonstrated that ToolChain* efficiently balances exploration and exploitation, outperforming state-of-the-art baselines by an average of 3.1% in planning tasks and 3.5% in reasoning tasks while requiring significantly less time−7.35 times and 2.31 times less, respectively. Not merely relying on retrieval algorithms, TOOLVERIFIER [85] addresses the challenge of generalizing language models to new tools with minimal demonstrations. It introduces a self-verification method that distinguishes between closely related tool candidates by self-posing contrastive questions during two critical stages: tool selection and parameter generation. In this approach, the language model generates synthetic, high-quality, self-generated data using Llama-2 70B, facilitating the construction of a robust evaluation function. Extensive experiments on four tasks from the ToolBench benchmark, encompassing 17 unseen tools, have shown that TOOLVERIFIER achieves an average improvement of 22% over few-shot baselines. This enhancement is particularly notable in scenarios where the distinctions between candidate tools are finely nuanced, underscoring the model's ability to assess and compare subtle differences between peer nodes effectively.

The tree-structured relationships between tools do not capture all invocation sequences, as some tools may have a precedence dependency on multiple other tools. For example, before controlling a robotic arm, it is necessary to calculate the optimal torque motor and simulation function, but there is no specific order between these tools. Directed edges in a graph can better capture the dependency relationships between tools than a tree structure. ToolNet [73] organizes tools into a directed graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ based on their relationships. Each node $\mathcal{N}$ represents a tool, and weighted edges $\mathcal{E}$ denote tool transition. Given a specified head node N_1 and a tail node N_2, the score on the edge E represents the contextual functional relevance between them:

$$\text{Score}(\mathcal{E}) = f(\mathcal{N}_1, \mathcal{N}_2) \tag{6}$$

In a tree structure, the state of $\mathcal{N}_2$ is uniquely determined by $\mathcal{N}_1$. However, in a graph, it could be influenced by multiple paths, such as $\{\mathcal{N}_1, \mathcal{E}_1, \mathcal{N}_2\}$, $\{\mathcal{N}_3, \mathcal{E}_2, \mathcal{N}_2\}$, and $\{\mathcal{N}_5, \mathcal{E}_3, \mathcal{N}_2\}$. The combined scores of these paths determine the decision to select $\mathcal{N}_2$:

$$\text{Select}(\mathcal{N}_2) = \text{argmax}\big(\text{Score}(\mathcal{E}_1), \text{Score}(\mathcal{E}_2), \text{Score}(\mathcal{E}_3)\big) \tag{7}$$

Starting from an initial tool node, an LLM navigates in the graph by iteratively choosing the next one from its successors until the task is resolved. Under the structure of the tool graph, the search algorithm shifts from trees to graphs, while the increase in the overall number of tokens consumed is limited.

Novel Frameworks In this subsection, we discuss some works based on the modifications in Fig. 3, which re-optimized and built a new framework without training LLMs. These frameworks include multi-stage pipelines and multi-agent frameworks. Sum2 Act [77] first utilizes a retriever to obtain relevant tools (or APIs) based on the correlation between users' instructions and descriptions or tools (or APIs). Then, it operates in two stages: the action proposal and summarization. In the action proposal stage, the router evaluates its progress, determining whether to conclude the task or propose a tool for further action. Subsequently, in the summarization stage, the state manager assesses the observations of these actions, updating the overall state accordingly. However, unlike user text queries, not all environments allow for the cost-free retraction of incorrect tool invocations. To fix this problem, SwissNYF [64] publishes the TOPGUN (Tool Orchestration and Program Synthesis for Generalizing over UNknown systems) framework unifies code generation, reasoning, and strategic tool planning designed for complex tasks. TOPGUN also verifies the execution plans and does so with exceptional efficiency in API cost, effectively addressing the limitation of API that it is inconvenient to be executed directly in the real environment or cannot be withdrawn after execution. The multi-stage pipeline was gradually redesigned as a multi-agent framework with role-playing in subsequent works. Wang et al [127] is the first to introduce role-playing in tool learning. He publishes Think-Plan-Execute (TPE), a framework that decouples the response generation process into three roles: Thinker, Planner, and Executor. Profiles define LLMs who play different roles. Specifically, the Thinker analyzes the internal status exhibited in the dialogue context, such as user emotions and preferences, to formulate a global guideline. The Planner then generates executable plans to call different conceptual tools (e.g., sources or strategies), while the Executor compiles all intermediate results into a coherent response. Similar to TPE, ConAgent [114] proposes another multi-agent cooperation framework. However, they additionally configured all tools with a monitoring tool, which is used to check whether these tools are in a normal state uniformly during the validation stage. They decompose the roles into Grounding, Execution, and Observing and use the Iterative Calibration (IterCali) method to calibrate themselves, utilizing the feedback from the tools when they fail to complete their actions.

3.2.2 Tool Documentation and Demonstrations

Tool documentation is an indispensable component, including function descriptions, parameter formats, open-source information, usage demonstrations, etc. Since LLMs that are not fine-tuned primarily rely on in-context learning, the examples provided in the demonstrations are of vital

importance [50]. Currently, most demonstrations are still based on text form, only Toubal [122] attempts to list tool-based visual examples when guiding VLMs in using tools for image classification. Most text-based toolsets are API interface repositories, which are far from mature, such as RapidAPI,[2] API-Bank,[3] and API-Bench.[4] For instance, not all components in the documentation are effective, and additional challenges include missing information, issues with format alignment, and redundant information. We summarize four normal issues in existing documentation:

- *Inconsistency* Massive tools from different sources often have inconsistent and diverse documentation formats, which makes it difficult to communicate consistent, universal explanations and examples. For instance, the documentation on RapidAPI at least includes API names, required parameters, optional parameters, API key, request URL, hostname, and protected words, but not all API providers offer demonstrations. In contrast API-Bench provides only functionality, API name, arguments, and demonstrations.
- *Redundancy* Tool documentation could encompass massive redundant and useless information, making it harder to grasp tool functionality and resulting in excessive token consumption in prompts. For example, RapidAPI implements additional steps and parameters during API calls to ensure secure access and cost calculation.
- *Incompleteness* Lack of methods for real-time tool updates and detection. APIs that involve time-sensitive information often face issues with updates to the method of invocation or with the format of the feedback results no longer compatible.
- *Impracticality* The essence of API calls is to write program interfaces and input the corresponding parameters. However, in more complex domains such as medical evaluation and robotic arms, relying solely on documentation makes it difficult for LLMs to fully understand how to use the tools. Moreover, demonstration examples may involve personal privacy, and generic demonstrations could face legal issues related to infringement.

From the perspective of historical development, AutoGPT [140][5] represents one of the earliest efforts to guide GPT-4 in selecting tools through prompts. Specifically, AutoGPT identifies intentions from user commands and constructs multiple prompts through iterative self-dialogue, enabling GPT-4 to select appropriate open-source projects for building code. For example, when a user requests the creation of a website, AutoGPT can automatically invoke the React framework to generate the front-end page. At this initial stage, documentation mainly consists of open-source project documentation and code-building demonstrations. Following AutoGPT, ResGPT [118] advanced the scenario toward more realistic applications by connecting LLMs with RESTful APIs, which adhere to the widely adopted REST architectural style for web service development. Subsequently, the research evolved from code generation toward invoking AI models. HuggingGPT [112] leveraged LLMs to integrate various AI models available on platforms such as Hugging Face to solve complex AI tasks. Specifically, upon receiving a user request, HuggingGPT first conducts task planning, selects appropriate models based on their documented functionality on Hugging Face, executes individual subtasks using the selected models, and finally summarizes the outcomes into a cohesive response. At this stage, documentation expanded to clearly describe the AI models inputs, outputs, functionalities, and methods for calculating invocation costs. As methods of model invocation matured, research attention shifted toward improving the efficiency of tool retrieval. ToolLLM [103] utilized RapidAPI to structure API directories into hierarchical tree structures, categorized with coarse-grained and fine-grained classifications, facilitating efficient API retrieval using tree-search algorithms such as Monte Carlo Tree Search (MCTS) [13]. However, since tree search algorithms depend heavily on node-state evaluations, Toolchain [170] proposed storing tool invocation sequences within the documentation and evaluating the current node state and invocation costs by comparing maximum common invocation sequences, further optimizing tool-selection processes. With the widespread adoption of the above approaches, issues inherent in tool documentation itself became increasingly evident, prompting researchers to focus more on the standardization and quality of documentation. To address inconsistency and redundancy, EASYTOOL [152] unified text compression techniques across documentation from diverse sources and constructed a standardized framework for tool descriptions. Additionally, EASYTOOL automated the process of filling parameters, verifying the accuracy of parameter descriptions, and validating correctness in tool invocation results. Addressing the incompleteness of tool documentation, especially in scenarios such as image and visual domains where demonstration data is often lacking, Hsieh et al. [50] proposed automated methods to rapidly integrate new tools along with their documentation, seamlessly expanding the toolset.As the practical usability of documentation emerged as a crucial bottleneck, especially for complex applications involving physical manipulators, relying solely on textual documentation became insufficient for LLMs to learn tool usage effectively. Consequently, recent research has shifted towards multimodal feedback

[2] https://rapidapi.com/.

[3] https://github.com/AlibabaResearch/DAMO-ConvAI/tree/main/api-bank.

[4] https://gorilla.cs.berkeley.edu/.

[5] https://github.com/Significant-Gravitas/AutoGPT.

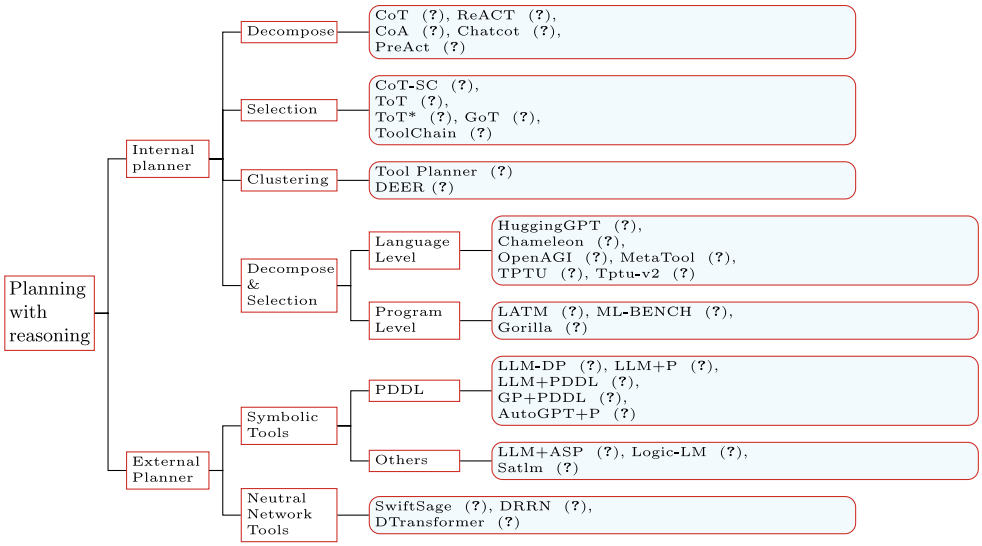

Fig. 6 The detailed categorization of all the work in the "Tool Planning" section is provided. However, due to space limitations, not all work shown in the figure will be discussed in detail in the subsections

methods. For instance, Incoro [168] enhances multimodal models with visual feedback mechanisms to help them better perceive and reason about the spatial positioning of robotic arms in dynamic environments. By incorporating real-time visual cues, the model can more accurately align its predictions with physical constraints in robotic manipulation tasks. Similarly, InRAP [49] focuses on interactive robotic planning by integrating electrical stimulation signals into the reasoning process. These signals, which are interpreted through external analysis tools, allow the system to estimate the magnitude and direction of forces applied to robot manipulators, thereby enabling more precise and context-aware action planning.

4 Tool Planning

Tools enable LLMs to access external knowledge, but there remain challenges for fine-tuned LLM agents (e.g., Toolformer [108]) to invoke tools in multi-step reasoning tasks, where inter-connected tools require holistic and efficient planning. This section will discuss tool usage planning methods, which are categorized based on whether they invoke external planning and reasoning tools. All the work in this section is presented in Fig. 6.

4.1 Internal Tool Planner

Tools usage planning methods without external tools primarily operate on three main principles: decomposition, selection and clustering.

Decomposition Task decomposition involves breaking down a complex task into several relatively simple subtasks and then planning the order in which these subtasks will be processed. Ideally, each subtask requires only one tool invocation, and this decomposition process occurs multiple times throughout the problem-solving process. The decomposition process can be formulated as follows:

$$\begin{aligned} &g_0, g_1, \cdots, g_n = \text{decompose}(\mathcal{LLM}, g; \Theta, \mathcal{T}); \\ &p^i = (t_0^i, t_1^i, \cdots t_m^i) = \text{sub-task}(\mathcal{LLM}, g_i; \Theta, \mathcal{T}). \end{aligned} \tag{8}$$

where $g, \Theta, \mathcal{T}, t^i$ represent the goals of output, the parameters of the LLM, the whole task and the tools. The key aspects of decomposition include the granularity of decomposition, the accumulation of errors, and the execution sequence. The granularity of decomposition is a critical prerequisite [145] for task resolution. Finer granularity may lead to resource wastage, as multiple sub-tasks could be efficiently handled by a single tool. Conversely, coarser granularity in subtasks may not contribute to reducing the complexity of the task, potentially hindering effective problem-solving. If the task relies on the outcomes of each step, such as Chain-of-Thought [134], errors generated in earlier steps can accumulate and magnify in subsequent steps. This problem is especially pronounced when solving mathematical equations [23], where both the process and the outcome require consistent accuracy. The execution sequence for subtasks depends not only on planning methods but also on tool retrieval and the evaluation of feedback from the tool invocation. In detail, CoT [134] initially decomposes a task into multiple sub-tasks using prompts, but this approach relies heavily on the detailed context and few-shot examples. ReAct [146] defines sub-steps in a task as a more detailed pattern of *'reasoning + action'* and then iteratively completes the entire task. CoA [34] fine-tunes the LLM with the goal of making reasoning chains with abstract placeholders. The placeholders do not affect LLMs reasoning flow and are subsequently infilled with specific knowledge retrieved from specialized tools to ground the final answer generations.

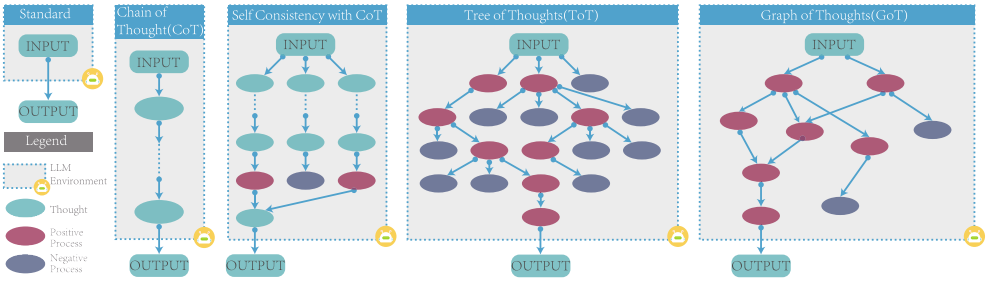

Fig. 7 Four typical reasoning methods are based on selection and decomposition. CoT decomposes a single task into multiple subtasks. CoT-SC performs the task multiple times and improves the answer quality through comparing and selecting. ToT and GoT use both decomposition and selection

Selection The selection of solutions primarily involves choosing an overall plan and selecting specific tools for subtasks, the key focus of which is on the evaluation methods. For overall plan evaluation, several methods adopt a strategy of generating and evaluating multiple complete reasoning trajectories before committing to action. Notably: CoT-SC (Chain-of-Thought Self-Consistency) [132] generates multiple reasoning paths and selects the final answer by majority voting among them, under the assumption that correct reasoning paths tend to converge. ToT (Tree of Thoughts) [79] explores a tree-structured search space of intermediate thoughts, where each branch corresponds to a partial reasoning path. It applies lookahead and value-based pruning strategies to select the most promising path. GoT (Graph of Thoughts) [8] extends this idea by modeling reasoning as a graph exploration problem, allowing multiple reasoning paths to merge and influence each other, enabling more flexible and efficient planning. These approaches share the idea of performing global planning and comparison of alternative trajectories, improving robustness and reasoning quality before step-by-step execution begins. As shown in Fig. 7, these steps are then assessed through human evaluation, comparative scoring, or objective metrics to select the optimal initial plan. For sub-step selection, ToolChain [170] combines the planning of tool usage with tree-search-based reasoning [145], which is particularly useful for procedural tasks [138]. The selection process can be formulated as follows:

$$\begin{aligned} P &= p_1, p_2, \cdots, p_n = \text{plan}(\mathcal{LLM}, g; \Theta, \mathcal{T}); \\ p^* &= \text{select}(\mathcal{LLM}, g, \mathcal{T}; \Theta, \mathcal{F}). \end{aligned} \tag{9}$$

where $\mathcal{F}$ represents the search strategies, such as *BFS, DFS* and *MCTS*. As discussed in Sect. 3.2, ToT [145] first structures the sub-steps of a task into a tree space and expands the sub-nodes using either a depth-first or breadth-first approach. During the evaluation of node states, backtracking is employed to avoid falling into local optima, and pruning techniques are used to reduce the search space. Building on this, ToT employs the MCTS algorithm [11] to identify the optimal planning path. By incorporating the A* search algorithm with task-specific cost function design, ToolChain [170] efficiently prunes high-cost branches that may involve incorrect actions, identifying the most low-cost valid path as the solution. Due to the special hierarchical structure of the toolset, the tool retrieval method becomes a core part of tool planning.

Additionally, there are a series of methods that simultaneously influence both tool invocation reasoning and tool planning strategies, organizing the usage of tools from different levels of abstraction.

At the high-level task planning level, some methods focus on orchestrating the sequence of tools required to solve multi-domain or complex mixed tasks: HuggingGPT [112] employs LLMs as task planners that assign subtasks to appropriate models hosted on HuggingFace. It decomposes complex instructions into structured workflows, enabling multi-model collaboration to handle tasks involving language, vision, and more. Chameleon [80] introduces a unified framework that supports dynamic tool routing, allowing the LLM to choose, combine, and switch between tools at runtime based on task context, effectively coordinating diverse tools for a single input. OpenAGI [36] proposes an open-agent framework that enables autonomous agents to select and invoke tools through iterative planning and feedback, supporting open-ended task solving across domains such as code, vision, and knowledge retrieval. MetaTools [54] focuses on building a meta-execution layer, where the LLM acts as a high-level controller to reason about what tool to use, when, and how, especially under dynamic or previously unseen toolsets.

At the programmatic or API integration level, some approaches aim to synthesize tool usage into structured programs or scripts, enabling more fine-grained procedural task automation. LATM (Large Action Model) [12] treats tool usage as part of an action space, where the model generates executable action programs by composing APIs to complete complex real-world tasks. ML-BENCH [75] proposes a

benchmark that evaluates LLMs's ability to plan and execute machine learning workflows, such as building, training, and evaluating models, by composing code snippets and API calls. Gorilla [98] fine-tunes LLMs to generate API calls grounded in open-source API documentation, enabling accurate tool usage in procedural tasks like configuring training pipelines or interacting with cloud services.

Clustering Clustering-based planning is designed to address situations where the tools within a toolset are disorganized and lack any categorical structure. Tool-Planner [76] categorizes tools into toolkits with similar or identical functionalities by clustering tool embeddings generated by SimCSE. The setting of toolkits allows for the thorough exploration of tools along a planning path, ensuring that each node is fully utilized and maximizing the information from that path. This framework enables task planning and tool invocation based on toolkits, addressing the inefficiencies in planning found in previous approaches. Like Tool-Planner, DEER [42] constructs the tool-usage samples with multiple decision branches via an automatic generation pipeline, thereby helping the decision-making abilities of LLMs. In addition to this, they design sampling methods for unclassified toolsets to classify them into fixed categories. Specifically, they define all tools $\mathcal{T} = \{t_i\}_{i=1}^{N}$, obtain the vectorized representation of tool $e(t)$ by embedding the tool's description d, i.e., $e(t) = \text{embedding}(t(d))$, and then the K-means algorithm is used to categorize all tools into m clusters.

4.2 External Tool Planner

Challenges will arise when internal tool planners confront environments featuring intricate constraints, such as mathematical problems or satisfiability issues. To tackle these challenges, various methods integrate LLMs with external planning tools, which are classified into symbolic and neural planners based on the tool categories [53]. The external tool planning process can be formulated as follows:

$$\begin{aligned} h &= \text{formalize}(\mathcal{LLM}, g; \Theta, \mathcal{T}); \\ p &= \text{plan}(\mathcal{LLM}, g, h; \Phi). \end{aligned} \tag{10}$$

where Φ denotes the external planner module, and h represents the formalized information. Due to the poor interpretability of neural networks and their strong dependence on data, more work has focused on external symbolic planners, as they can almost losslessly translate any data into symbolic form.

Symbolic planners Symbolic planners have served as a fundamental component in the fields of tool planning for several decades. These methods, which rely on well-established symbolic formalized models like PDDL models [2], utilize symbolic reasoning to determine optimal paths from initial states to desired goal states. LLM+PDDL [115] is the first to employ the PDDL language to structure tasks, adding a manual verification to identify and address potential flaws in the PDDL models generated by the LLM. Additionally, by capitalizing on LLMs' semantic understanding and coding abilities, LLM+P Liu et al. [69] translates problems into PDDL prompts that are inputted into the LLMs. Subsequently, after obtaining a formalized description, it further employs the Fast-Downward[6] solver for the planning process. Building upon LLM+P, LLM-DP [24] is specifically designed for dynamic interactive environments. Upon receiving feedback from the environment, LLM processes the information, formalizes it into PDDL language, and then employs a BFS [60] solver to generate a plan. In addition to solving planning tasks under the former closed-world assumption, AutoGPT+P [10] can further handle planning with incomplete information, e.g., tasks with missing objects by exploring the scene, suggesting alternatives, or providing a partial plan.

The above works focus on converting natural language into PDDL. Other symbolic representations can significantly improve problem-solving efficiency in mathematics and logical reasoning scenarios. LLM+ASP [143] converts descriptions in natural language by LLM into atomic facts, thereby translating tasks into ASP (Answer Set Programming) problems, which are then addressed using specialized external tools. Similarly, Logic-lm [94] transforms described problems into first-order formal logic (FOL [7]). It then utilizes external logic tools to discern the logical sequence of the problem and find the solution plan. Satlm [150] transforms theorem-proving tasks, which originally required programmatic representation, into satisfiability problem representations(SAT), using an automated theorem prover to generate final answers for these declarative task specifications. Its performance far surpasses that of methods combining chain-of-thought and programs.

Neural planners. Neural planners are advanced deep learning models that are trained on extensive planning data using reinforcement learning or imitation learning.

Well-trained neural planners demonstrate exceptional planning capabilities within tool-learning domains and achieve superior planning efficiency due to their compact parameter sizes. However, these smaller models often struggle with complex and rare problems where training data is limited, primarily due to their inadequate generalization abilities. Recent research explores ways to combine the strong generalization of large language models (LLMs) with the planning efficiency of smaller neural planners in tool invocation tasks. In detail, DRRN [47] conceptualizes the planning process as a Markov Decision Process, utilizing reinforcement learning to train a policy network that develops a sophisticated decision-making

[6] https://github.com/aibasel/downward/tree/release-22.12.0.

model. Similarly, the Decision Transformer (DT) [16] enhances a transformer model with the capability to emulate human decision-making behaviours by leveraging planning data. SwiftSage [68] takes inspiration from cognitive psychology's dual-process theory, distinguishing between fast and slow thinking in the planning process. Fast thinking involves instinctive, rapid responses based on learned experiences, while slow thinking entails careful reasoning and deliberation. In SwiftSage, a decision transformer model trained via imitation learning handles the fast-thinking component and quickly generates plans for routine tasks. When a planning error suggests a more complex problem, the agent switches to slow thinking, where the LLM performs deeper reasoning and strategic planning based on the current context. This dual-process integration has been shown to enhance planning efficiency significantly.

5 Interaction with Environment

Prior work limited LLMs' perceiving tool-use ability to a single text query, which may result in ambiguity in understanding the users' real intentions. Meanwhile, the pure text-based perception of LLM makes it difficult to choose the right multi-modal tool when completing image generation tasks and image understanding tasks, even if the correct tools are available in the tool library [156]. For the given textual prompt, "*Generate a picture conditioned on the image. A professional athlete dressed in a football uniform, throwing his arms up in a cheer.*", LLMs struggle due to a lack of visual input and cannot specify the type of conditioned image. Recognizing this as an image generation task, LLMs still falter in recommending suitable pose- or scribble-conditioned text-to-image tools. LLMs are expected to eliminate that by perceiving the information in the visual- or auditory-grounded instructions. Next, we will discuss tool-learning works that expand multi-modal tools and inputs.

5.1 Multi-modal Inputs

To address the aforementioned challenges, MLLM-Tool [126] introduces a framework that processes visual- and audio-grounded instructions by incorporating multimodal encoders and open-source large language models from Huggingface. Specifically, it adopts ImageBind [38] as the primary multimodal encoder, enabling the system to align and interpret inputs across various modalities effectively. This design enhances the model's ability to accurately invoke and utilize external multimodal tools, such as those for image or audio understanding, thus improving task performance in complex, tool-augmented multimodal scenarios. Specifically, it first utilizes ImageBind encoders with fixed weights for feature extraction. These features are then aligned with the LLM's feature space through a linear projection layer. Then, Low-Rank Adaptation (LoRA) [51] is implemented to fine-tune an open-source LLM. Furthermore, a benchmark MLLM-Bench is built to evaluate MLLMs' awareness and selection ability regarding external tool usage with text-ambiguous queries. This benchmark offers content-aware multi-modal instructions and one-to-many instruction-answer pairs; please refer to Sect. 7.1 for more information.

LLaVA-Plus [72] maintains a skill repository that contains a wide range of vision and vision-language pre-trained models (tools) and can activate relevant tools, given users' multimodal inputs, to compose their execution results on the fly to fulfill many real-world tasks. Unlike MLLM-Tool's SFT method, LLaVA-Plus is based on pre-training, and the pre-training data covers many tool use examples of visual understanding, generation, external knowledge retrieval, and compositions. This enables it to have better generalization capability across unseen multimodal toolsets.

5.2 Multi-modal Tools

Multimodal toolsets may simply extend the application scenarios for tools, but the model invoking the tools may still be LLMs. This is because some toolsets, although multimodal (e.g., text-to-image or text-to-video), still take text as input.

CLOVA [35] is a visual assistant that updates its tools within a closed-loop learning framework for better adaptation to new environments. This framework has a multimodal global–local reflection scheme capable of identifying tools needing updates. Specifically, if a task is not solved correctly, this reflection scheme uses LLMs to generate critiques, identifying which tool needs to be updated. Zhang et al. [157] propose Graph-ToolFormer, a novel framework designed to teach large language models (LLMs) how to utilize external graph reasoning APIs through self-supervised prompting. Specifically, the model is trained using ChatGPT-generated augmented prompts that simulate realistic query scenarios involving graph-related reasoning tasks. This enables the LLM to learn how to identify appropriate situations for tool invocation, formulate API calls, and interpret the returned results. In addition to the self-instruction mechanism, Graph-ToolFormer makes a notable contribution by extending the toolset from unimodal (text-only) tools to multi-modal settings. This allows the framework to handle complex reasoning problems that require cross-modal understanding, such as linking visual data with graph-structured knowledge or interpreting tabular data within

a knowledge graph context. Through this enhancement, the framework bridges the gap between general-purpose LLMs and specialized symbolic reasoning systems, paving the way for more accurate and explainable multi-hop inference. In the financial field, FinAgent is a multimodal foundational agent enhanced for financial transactions. It utilizes additional multimodal tools in its market intelligence module to process various data types numerical, textual, and visual- to analyze financial markets accurately. Its dual-layer reflection module adapts quickly to market dynamics. It integrates a diverse memory retrieval system, enhancing the agent's ability to learn from historical data and improve decision-making processes. Modeling Collaborator [157] leverages multimodal tools to automatically perform image classification tasks, thereby significantly reducing the reliance on human annotators and lowering the costs typically incurred through crowdsourcing efforts. By integrating tool-use capabilities into the LLM framework, the system can independently invoke appropriate vision models when faced with visual inputs, enabling more efficient and scalable annotation workflows.

6 Applications for Tool Learning

This section primarily discusses the application of tool-learning agents in various specialized fields. From a development perspective, the application of tools in software code generation, AI model invocation, data analysis, and mobile/PC domains is particularly important. At the same time, work in these fields has experienced explosive growth over time.

6.1 Data Science

The application of tool learning agents in data science mainly focuses on databases and data analysis, with most of the data analysis work targeting finance. Additionally, bioinformatics applications will be discussed separately in a later section.

Database In data science, many tools are utilized [14, 110, 163], among which databases are paramount, offering four fundamental functionalities: insert, update, delete, and query. LLMDB [167] has enhanced the model's capability to invoke these tools by embedding domain knowledge of databases as an external knowledge base and fine-tuning the general model with user-sourced data. Meanwhile, Liu et al. [71] has improved LLM caching reuse and eliminated redundant inference requests by reordering key-value pairs within tools, thereby reducing the cost associated with data queries. Pedro et al. [99] has reduced the vulnerability of databases to injection attacks by augmenting the Langchain framework [121] with tool extensions. Gu et al. [41] design a middleware that serves as a layer shielding the LLM from environmental complexity, such as knowledge bases(KBs) and databases Han et al [44], Chen et al. [15], Yang et al. [139]. These tools can aid in the proactive exploration within the above massive environments.

Finance In specialized domains such as finance, where data is often highly heterogeneous (e.g., numerical reports, regulatory text, market trends) and precision is critical, recent work has explored the use of tool-augmented language models to improve task-specific reliability and adaptability. For example, Theuma and Shareghi [120] fine-tune LLaMA-2-13B-Chat using supervised learning to create a dual-role architecture, enabling the model to function as both a "task router" and a "task solver". The task router is designed to dynamically analyze each incoming financial query and decide whether the question should be handled internally by the LLM or delegated to an external tool from a curated toolset (e.g., a stock data API, risk calculator, or tax regulation parser). Meanwhile, the task solver component is trained to interpret tool-related instructions, construct appropriate API calls, and correctly interpret their outputs. This two-stage architecture ensures accurate tool selection and execution in the context of complex, real-world financial workflows. Similarly, Zhao et al [160] focus on domain-specific adaptation by fine-tuning open-source LLMs with finance-oriented corpora and integrating them with data analysis tools. Their system targets core areas such as investment management, wealth preservation, and tax optimization domains that require not only factual accuracy but also nuanced interpretation of regulations and strategic planning. By combining language modeling with structured financial tool usage, the model can assist in making context-aware financial decisions, thereby improving the reliability and transparency of AI-driven advisory systems. These works collectively demonstrate how tool-augmented LLMs can be tailored to meet the demands of specialized, high-stakes domains, offering both improved interpretability and actionable outputs.

6.2 Software Engineering

In software engineering, tools are primarily focused on 1) code understanding and generating; 2)error analysis. For code understanding, an IDE plugin [92] enhances code comprehension by integrating API interpreters directly within the IDE compiler. Extensive user testing has shown that this system effectively outperforms traditional web-based queries. At the same time, a copilot evaluation tool [3] is used to assess the data and tools of LLM-guided IDE interactions, covering various programming scenarios and languages. For error analysis, Roy et al [105] has enabled LLMs to call upon external diagnostic services for errors, significantly

enhancing the efficiency of Root Cause Analysis (RCA) automation. GitAgent [82] is an error analysis agent capable of achieving the autonomous tool extension from GitHub, which follows a four-phase procedure to incorporate repositories, and it can learn human experience by resorting to GitHub Issues/PRs to solve problems encountered during the procedure.

6.3 Robot Control

The application of Large Language Models (LLMs) [21, 66, 128, 136] in robotic control has gained increasing attention, particularly in the manipulation of mechanical arms [129], where multimodal feedback such as visual, textual, or electrical signals plays a crucial role in enhancing control precision and adaptability. The development of LLMs in this domain can be broadly divided into two main categories:1. Simulation-based control within virtual environments, where robot actions are trained and evaluated in software simulators; and 2. real-time control of physical robotic arms, where LLMs directly or indirectly influence actions in the physical world.

However, the majority of existing research tends to emphasize high-level decision-making and task-level planning rather than fine-grained control. This is largely due to LLMs's limited capacity for interpreting low-level signals, such as electrical or force feedback, and their inability to precisely compute physical simulation parameters required for accurate low-level control. The early work by Hori et al. [49] marked a key milestone by applying LLMs in VirtualHome [100], where language models were used to control virtual agents performing object relocation tasks in household-like environments. This demonstrated the feasibility of integrating LLMs with structured, simulation-based control pipelines. Building upon this, researchers such as Bhat et al. [9] and InCoro [168] extended the scope to real-world robotic systems. They utilized a plan action reflection framework, allowing the LLM to reason over structured plans, execute motor actions, and revise strategies based on multimodal feedback. These feedback signals often include visual data from cameras or depth sensors and, in some cases, proprioceptive signals like joint positions and torques. Further advancing the field, CognitiveOS [81] proposes a cross-platform interactive architecture for robot control. It aims to build a generalizable system-level operating interface that allows mechanical arms to be controlled across multiple simulation and physical environments, supporting modularity, extensibility, and more interactive feedback loops. Together, these works highlight both the promise and limitations of current LLM-driven robot control systems, showing that while high-level planning is well supported, challenges remain in bridging the gap to precise, low-level mechanical execution.

6.4 Biomedical

In biology, tool learning primarily focuses on retrieving and identifying biological information. In contrast, in medicine, it is chiefly used for diagnosing patient conditions and assessing the medical standards of populations.

Medicine In the healthcare and biomedical domain, the integration of external tools into large language models (LLMs) has shown great promise in enhancing clinical decision-making, automating medical workflows, and improving data processing efficiency. AgentMD [59] exemplifies this trend by integrating clinical calculators directly into LLMs. The system can automatically identify and invoke appropriate RiskCalcs tools based on any free-text patient description, enabling accurate risk stratification and assessment of patient characteristics. This reduces the need for manual tool selection and supports more intelligent, patient-specific reasoning. Similarly, Biotreasury [162] focuses on biological data management. Coupling LLMs with information retrieval tools significantly improves the efficiency of biological classification tasks and accelerates the construction of biological databases, which are crucial for genomic research and drug discovery. Imrie et al. [55] demonstrate that LLM-based clinical systems can serve as a flexible interface between clinicians and digital medical tools, enabling models to call external resources such as medical calculators, drug databases, or EHR summarizers to assist in evidence-based care. On the imaging side, Meskó and Topol [86] explore the use of LLMs equipped with image-processing tools to analyze medical imaging results, allowing models to extract contextual information from visual inputs and integrate them into comprehensive health assessments. Going further, EHRAgent [113] introduces a code-generation interface within the LLM framework, enabling the model to autonomously write and execute code for multi-tabular reasoning tasks in electronic health records (EHRs). This supports advanced queries such as time-based trend analysis or patient cohort selection, which typically require complex data manipulation across multiple clinical tables. These systems collectively demonstrate how tool-augmented LLMs are reshaping clinical AI by moving beyond pure language understanding toward actionable, tool-driven decision support in real-world healthcare environments. They enhance the LLM agent by incorporating long-term memory, which allows EHRAgent to effectively select and build upon the most relevant successful cases from past experiences. At the same time, newly successfully predicted diagnostic records are continuously added to the database.

Biology Only a very limited number of works apply tools to bioinformatics retrieval. For example, GeneGPT [58] teaches LLMs to use the National Center for Biotechnology Information (NCBI) Web APIs to answer genomics questions. Specifically, they prompt Codex to solve the

GeneTuring tests with NCBI Web APIs by in-context learning and an augmented decoding algorithm that can detect and execute API calls.

6.5 Web and Mobile

Mobile devices and personal computers are crucial interfaces for LLMs because they represent the most significant touchpoints for exchanging personal information. The work of tool learning agents on mobile and PC platforms can be divided into two categories: (1) agents that optimize based on offline data and perform well on corresponding datasets and apps and (2) agents that are based on online frameworks and can run directly on real mobile devices.

Online Frameworks Android-Env [123] first identifies this potential and develops a platform that enables Large Language Models (LLMs) to interact with mobile phones, capturing action feedback through logs. At this time, these LLMs are primarily focused on mastering mobile gaming. Building on this, Mobile-Env [155] advances the environment by replacing log-based feedback with HTML-structured text, extending the LLM tasks to encompass browser-based information retrieval, thus bringing LLMs' use of mobile technology within a step of everyday scenarios. Subsequently, MobileBench [26] builds a general mobile agent framework based on simulators, significantly reducing the cost of deployment in real environments.

Offline Agents Early efforts such as Rico [25] laid the foundation for large-scale GUI datasets by collecting over 700,000 mobile and desktop page data entries from Google applications. These data were organized into page sequences guided by task instructions, enabling foundational research on user interface modeling and layout understanding. However, due to the rapid evolution and iteration of modern applications, the static nature of Rico has limited its applicability in more dynamic, real-world mobile scenarios.

To address this, Auto-UI [159] leverages more realistic and diverse data sources, notably from Android in the Wild (AitW) [104], and constructs a collection of routine app testing tasks via crowdsourced annotation. This enables the dataset to capture a wider variety of interaction patterns across different Android devices and app versions, better reflecting the complexity of real-world usage. Given the inherent limitations of purely text-based interfaces in capturing and reasoning over complex GUI states, several recent works have shifted toward multimodal representations that integrate vision and action signals. For example, AppAgent [144] proposes a unified framework that grounds instructions into executable actions by interpreting GUI screenshots along with element metadata. CogAgent [48] introduces a cognitively-inspired approach, encoding UI structures and action semantics into a shared embedding space to improve alignment between intent and interface. Ferret-UI [151] extends the Ferret framework to GUI scenarios, supporting visual grounding through pixel-wise element understanding and action prediction. MobileAgent [130] emphasizes task planning and multi-step action execution, using visual observations to guide complex navigation and manipulation in mobile apps. Building on this trend, COCO-Agent [83] focuses exclusively on mobile page understanding. It is pre-trained on a large-scale corpus of mobile interfaces and fine-tuned for GUI navigation tasks. As a result, it achieves state-of-the-art performance in path planning accuracy and page-level click prediction, demonstrating the advantages of domain-specific pretraining for mobile GUI agents.

7 Evaluation

In this section, we list benchmarks and widely used evaluation indicators used to evaluate the tool-learning capabilities of Agents. The current toolset consists of APIs for AI model invocation, real-time information query, and functional tools such as scientific calculator, date conversion, and exchange rate calculation. Initially, toolsets and test benchmarks evolved from web-based page construction to AI model invocation and real-world scenarios such as daily information queries, retrieval, and classification across various domains. The latest benchmarks now target complex scenarios with noisy and ambiguous instructions. The evaluation metrics are composed of manual assessment, advanced model evaluation, and calculation of objective indicators.

7.1 Benchmarks

This section will systematically introduce these benchmarks according to task complexity, tool type, and task objective. This classification method is used primarily because tasks and toolsets are closely related. All the benchmarks are listed in Table 2.

7.1.1 Task Complexity

The complexity of tool-learning tasks can range from simple single-step API calls to complex multi-step tool planning and execution. Based on the number of steps and the tools involved, the related benchmarks can be divided into two categories: (1) Single-step invocation and simple tasks; (2) Multi-step invocation and complex tasks.

Single-step invocation and simple tasks. In these benchmarks, one task typically involves only one tool, or even if multiple tools are involved, their invocation is relatively simple. The primary focus is to test whether the model can correctly select and invoke the appropriate tool based on the instructions. API-Bank [67] is one of the earliest tool-learning benchmarks, classifying tasks into single-API

Table 2 These datasets are specifically designed to measure the ability of model tools to learn and use

Dataset	Open source	Domains	Scale	Annotations
API-Bank [67]	–	1008 domains	2211 APIs	Train, test
APIBench [98]	HuggingFace&Torch Hub	Machine learning	1645 APIs	Train, test
ToolBench [138]	RapidAPI	Real world	16k APIs	Fine-tune, test
ToolQA [169]	–	Real world	13 APIs	Pretaining, fine-tune, 1530 test
ToolAlpaca [119]	APIs	Real world	400 APIs	Train, test
RestBench [118]	TMDB, Spotify	Real world	94 APIs	Development, test
ToolBench [103]	RapidAPI	Real world	16k APIs	Train, test
TOOLE [54]	OpenAI	Real world	390 tools	Test
MenatQA [135]	TimeQA [17]	Temporal domain	1 temporal tool	1,448,857 train &2853 test
MLLM-Tool [126]	HuggingFace	Real world	932 APIs	Train, test
UltraTool [52]	Chinese, English	Real world	2032 tools	Test
ToolTalk [31]	–	Physical world	28 tools	178 SFT data, test
ToolEyes [149]	SerpAPI	Physical world	600+ tools	Train, test
Stabletoolbench [43]	RapidAPI	virtual world	16k APIs	Fine-tune, test
RoTBench [148]	ToolEyes [149]	Physical world	568 APIs	1800+ SFT data, test
InjectAgent [154]	LM-Emulated Sandbox [107]	(IPI) Risk	17 API &62 attacker tools	1054 test
MLLM-Tool [126]	HuggingFace	AI model	932 APIs	45k train, 2.2k test
T-Eval [19]	RapidAPI	Real world	16k+15 APIs	23,305 test
Noisy-ToolBench [131]	RESTful APIs	Real world	100 APIs	1000 test

Domain is the vertical domain involved in API calls and returned knowledge; open source is the source of API calling functions and documentation. Scales represent the number of APIs or tools in the toolset, while annotations include how the data set is divided

or multi-API calls. In particular, it divides tasks into the following three categories and evaluates their abilities in different aspects: (1)Few APIs, Single Call or Multi Calls (Call): when the API is known, the ability to call the correct API based on given instructions; (2)Many APIs, Single Call (Retrieval+Call): the ability to retrieve and call the correct single API when the API is unknown; (3)Many APIs, Multi Calls (Plan+Retrieval+Call): the ability to specify plans, retrieve and call the correct multiple APIs when the API is unknown. Additionally, it defines tool invocation as a multi-turn dialogue to reduce the task's difficulty. The LLM generates the test instructions automatically, while the answers are manually annotated. APIBench [98] generated 16,460 API invocation instructions by collecting APIs from HuggingFace, TensorFlow, and Torch Hub. This benchmark focuses on the correct invocation of individual APIs, particularly those from the machine learning domain. Similar to API-Bench, ToolBench [103] provides a real API set, with tasks divided into single-tool invocation (I1) and multi-tool invocation (I2, I3). The I1 task tests the model's ability to invoke a single tool based on instructions, assessing whether the model can execute the correct invocation when it has seen the tool but not the specific instruction.

Multi-step invocation and complex tasks. Single-tool benchmarks are mainly used to test whether LLMs can solve simple, non-planning tasks while assessing their ability to fill in parameters correctly. Real-world tasks involve multi-step, multi-tool invocations, requiring models to plan, select, and coordinate multiple APIs to complete complex tasks. As a result, benchmarks involving multiple tools are developed. RestBench [118] is derived from ToolBench but focuses more on complex task scenarios. Specifically, it covers two complex real-world scenarios IMDB movie database and Spotify music player requiring the model to perform multi-step invocations across multiple APIs and provide the correct solution paths. This benchmark evaluates the invocation of individual APIs and tests task planning and execution across multiple APIs. UltraTool [52] extends tools from the virtual scenarios of RESTful collections to real-world scenarios. It covers 2032 tools across 22 domains and emphasizes real-world complexities, demanding accurate, multi-step planning for effective problem-solving. ToolTalk [31] evaluates LLM performance in complex tasks by simulating an environment with 7 plugins and 28 tools. This benchmark is designed with multi-step tool operations, simulating action tools that can affect the external world. In the test scenarios, once a tool is invoked, it cannot be retracted. Unlike the parameter-focused dialogue in API-Bank, ToolTalk defines the entire problem-solving process as a multi-turn dialogue to reduce complexity. It contains a total of 78 dialogues and 178 turns in total.

7.1.2 Toolset Type

The toolsets in benchmarks can be derived from virtual APIs or real APIs, and at the same time, the tasks in these benchmarks may involve multiple tools. However, due to the distinct characteristics of their toolsets, we will discuss them in this subsection.

Virtual Tools Some benchmarks use virtual APIs to create a more stable evaluation environment, avoiding the impact of real API fluctuations on evaluation results. For instance, StableToolBench [43] is a benchmark evolved from ToolBench [103], introducing a virtual API server and a stable evaluation system. The virtual API server in this benchmark includes a caching system and an API simulator to mitigate the issues caused by API state changes, ensuring the stability of tool evaluations.

Real Tools Some benchmarks directly use real-world APIs for testing, enhancing the realism of the tasks and their practical applicability. Toolformer [109] is the first innovative AI model designed to use several specialized tools, such as a web browser, a code interpreter, and a language translator, within a single framework. From the web to the real world, ToolBench [103] collects 16,464 real and usable APIs through the RapidAPI Hub and generates relevant instructions using ChatGPT. This benchmark provides a more challenging evaluation of a model's ability to use tools through real API invocation tasks. To address the issue of insufficient examples in ToolBench, ToolAlpaca [119] collects over 400 real-world APIs covering 50 different categories, constructing 3938 tool usage instances. ToolEyes [149] provides around 600 tools. This benchmark meticulously examines seven real-world scenarios—text generation, data understanding, real-time search, application manipulation, personal life, information retrieval, and financial transactions, and analyzes five dimensions crucial to LLMs in tool learning: format alignment, intent comprehension, behavior planning, tool selection, and answer organization. Additionally, single-modal tools cannot handle multimodal tasks like "text-to-image" generation. As a result, benchmarks involving multimodal tasks have become a current hotspot. MLLM-Tool [126] scrapes 263,945 models from the HuggingFace platform and, after filtering, retained 932 high-quality model APIs. This benchmark evaluated the invocation capabilities for text, image, audio, and video APIs, serving as a cross-modal, multi-task evaluation platform.

7.1.3 Special Scenarios

In addition to the mainstream scenarios mentioned above, some benchmarks focus on evaluating the model's robustness and security, while others concentrate on the LLM's ability to handle ambiguous instructions.

Robustness and Security These benchmarks evaluate the robustness and security of LLMs in tool usage by introducing different levels of external noise or potential attacks. RoTBench [148] assesses the robustness of LLM tool-learning models through five external environments with varying noise levels (Clean, Slight, Medium, Heavy, Union). It tests the model's noise resistance in three key stages: tool selection, parameter recognition, and content generation. Unlike RoTBench, InjectAgent focuses on constructing a more hostile environment. InjectAgent [154] is designed to evaluate the vulnerability of LLM agents integrated with tools against IPI attacks. This benchmark contains 1054 test cases, covering 17 user tools and 62 attack tools, focusing on assessing the model's robustness and data security when facing malicious attacks.

Ambiguous Instruction Handling Ambiguous instructions often lead to errors in tool invocation. These benchmarks primarily test whether the model can recognize and handle vague or unclear user instructions. For this purpose, Noisy-ToolBench [131] tests the model's ability to handle ambiguous or unclear instructions. This benchmark includes 100 RESTful APIs and 1000 ambiguous instructions manually constructed by experts, testing whether LLMs can identify these ambiguous instructions and proactively prompt the user for clarification.

7.1.4 Others

In addition, some benchmarks are difficult to categorize. For example, ToolQA constructs the data directly as tool-specific question-and-answer tasks. ToolQA [169] contains data from 8 different fields, each with two types of questions, easy and hard. The detailed statistics of the dataset are as follows: (1) Temporal: Flight and Coffee data, containing 408,318 and 5746 records. (2) Spatial: Yelp and Airbnb data, containing 150,346 and 102,599 records. (3) Mathematical: GSM8 K [23], containing 100 records. (4) Social: DBLP [90], containing 553,320 records. (5) Scientific: SciREX [56], containing 438 records. (6)Personal: Agenda containing 10,000 records. TOOLE [54] contains 390 tools whose names and descriptions were retrieved from the OpenAI plugin list, which can evaluate whether LLMs have tool usage awareness and can correctly choose tools. This benchmark includes four subtasks from different perspectives in tool selection, including tool selection with similar choices, tool selection in specific scenarios, tool selection with possible reliability issues, and multi-tool selection. MenatQA [135] contains three time-sensitive factors: scope, order, and counterfactual. A total of 2853 samples are used to evaluate LLMs' time understanding and reasoning capabilities. Furthermore, this work undertakes a preliminary investigation into potential improvement strategies by devising specific prompts and leveraging external tools.

Table 3 The table records commonly used evaluation indicators for tool learning

Metric	Category	Formula	Annotations
Pass Rate [103]	Result Check	Eq. 11	Model Generation
Pass Rate* [29]	Result Check	Eq. 12	Model Generation
Win Rate [103]	Process Check	Eq. 13	Model Generation
Accuracy [126]	Result Check	Eq. 14	Objective indicators
Plan Score [19]	Plan Check	Eq. 19	Model Generation
ROUGE-L	Plan Check	Eq. 18	Objective indicators
ASTSM [98]	Process Check	–	Objective indicators
CSR [54]	Result Check	Eq. 21	Objective indicators
Hallu Rate [126]	Result Check	Eq. 22	Objective indicators
Levenshtein Dis [89]	Plan Check	Eq. 20	Objective indicators
Average-Steps [155]	Result Check	–	Objective indicators
Recall [126]	Result Check	Eq. 15	Objective indicators
MD [52]	Result Check	–	Model Generation

Metric records the name and source of the indicator; Category contains two categories, marking whether the indicator is the evaluation process or the final result; Formula is the calculation formula of the indicator (If exists); The annotation contains an explanation of the indicator calculation methods, including the following three: direct calculation of objective indicators, model evaluation, and manual evaluation

Some people also evaluate the finetuned models on other non-tool-specific general datasets, which require multi-step reasoning, such as mathematical reasoning [23, 87, 97], and Wikipedia(WikiQA) [65, 142]. Tool invocation evaluation sets exist as either a subset or an enhancement in these benchmarks.

7.2 Metrics

In this section, we categorize the evaluation metrics into the following four types based on dimensions and objectives: (1) evaluation of task completion, (2) fine-grained evaluation of output quality, (3) evaluation of tool selection and invocation ability, and (4) evaluation of robustness and error-handling capabilities. All the metrics are listed in Table 3.

7.2.1 Task Completion

The metrics in this subsection focus on whether the model can successfully complete the task and whether the task execution meets expectations. These metrics are used to evaluate the overall task completion by the model, as well as the completion of each step in multi-step tasks. *Pass Rate* is introduced in ToolLLM. It measures the executability of LLM in terms of calling tools, given a max-length. It calculates the proportion of instructions that are successfully completed under this limit. The following equation can calculate this metric:

$$PassRate = \frac{\#(Non-solvable) + \#(Solved)}{\#(Non-solvable) + \#(Solved) + \#(Unsolved)}. \tag{11}$$

In the Toolbench, each model is required to output the"Finish with Final Answer"action after completion, and the percentage of the output of this action is directly counted after excluding"false positives". Xu et al [138] also uses this metric to evaluate the ability of LLM. To address the limitations inherent in ToolLLM [103] evaluation protocol, researchers propose an alternative evaluation metric called Pass Rate* that aligns more closely with real-world scenarios. Specifically, they bypass the first evaluation stage of ToolLLM, which assesses the potential of candidate APIs in addressing query $\mathcal{Q}$, and only check those Qs that the tools can solve. The following equation can calculate this metric:

$$PassRate *= \frac{\#(Solved)}{\#(Solved) + \#(Unsolved)}. \tag{12}$$

The Win Rate is also introduced in ToolLLM, but unlike Pass Rate, which only measures whether instructions can be completed, Win Rate measures how well instructions are completed. It determines which of the two given solution paths is better. This requires providing a set of standard paths for comparison. In ToolLLM, this standard path is set as the output of GPT-4 in Toolbench. Win Rate needs to be evaluated multiple times to improve reliability, and the following equation can calculate it:

$$Acc = \sum_{i=1}^{n} \mathbb{1}(P_{pre}, P_{gold})/n \tag{13}$$

In addition, some general-purpose metrics can also be used to examine the tool invocation process. For example, *Accuracy* is used to calculate the API's accuracy by dividing the number of correct calls by the total number of calls. The testing set is divided into several subsets based on the three attributes. For a subset $\mathbf{S}$, supposing containing n instructions $\{x_1, x_2, x_3 \dots x_n\}$, where each instruction x_i matches an API list y_i, which consists of a list of M suitable APIs $[A_i^1, A_i^2, A_i^3, \dots, A_i^M]$. Thus, the collection of API lists corresponding to this instruction subset combined to a set $\mathbf{Y}$, $\{y_1, y_2, y_3 \dots y_n\}$. With these symbols defined, assuming the subset $\mathbf{S}$ yields an API set $\{z_1, z_2, z_3 \dots z_n\}$, output from our model, we can represent the accuracy of each mentioned subset using the following formula:

$$\text{Acc} = \sum_{i=1}^{n} \mathbb{1}_{z_i \in y_i}/n \tag{14}$$

Unlike simple one-to-one mapping relationships, using single-instance Accuracy is insufficient for evaluating the system's comprehension of the APIs relationships, especially in one-to-many mapping relationships. Recall is then used to calculate the repeated inference results. By examining the proportion of ground truth encompassed by the collective results of multiple inference results, marked as R_i^j, where i is the i^{th} instruction in the testing set and j represents the j^{th} inference. The corresponding metric calculation is expressed as follows:

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{(f_c(Set(R_i^1, R_i^2 \dots R_i^K | R_i^j \in y_i, j \in [1, K]))}{f_c(y_i)} \tag{15}$$

where K is the number of inference, f_c is the counting function.

7.2.2 Output Quality

Evaluating the quality of the model's output during tool invocation can be divided into two categories: (1) evaluating the quality of tool invocation planning and (2) evaluating the quality of parameter filling.

Plan Quality Recall-Oriented Understudy for Gisting Evaluation—*ROUGE-L* is a popular evaluation metric used in natural language processing, particularly for assessing the quality of text summarization and machine translation. It measures the longest common subsequence (LCS) between the generated text and reference text. This metric can also be used to evaluate the differences between predicted plans and standard plans. Assuming X is a reference plan sentence with length m, Y is a candidate plan sentence with length n, the calculation formula of ROUGE-L is as follows:

$$R_{LCS} = \frac{LCS(X,Y)}{m} \tag{16}$$

$$P_{LCS} = \frac{LCS(X,Y)}{n} \tag{17}$$

$$Rouge - L = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS} + \beta^2 P_{LCS}} \tag{18}$$

Plan Score [19] is a metric to evaluate the similarity between the predicted plan from the LLM $P^{pred} = [a_1^{pred}, a_2^{pred}, \dots, a_{n^{pred}}^{pred}]$ and the gold answer $P^{gt} = [a_1^{gt}, a_2^{gt}, \dots, a_{n^{gt}}^{gt}]$ from human annotators, the planning evaluator begins by computing a similarity matrix S. This matrix represents the similarity scores for all action pairs $(a_i = (tool_i, args_i), a_j = (tool_j, args_j))$ between the prediction and the golden answer:

$$S_{i,j} = \beta\sigma(tool_i, tool_j) + (1-\beta)\sigma(args_i, args_j).$$

After getting the similarity matrix S, the Longest Increasing Subsequence (LIS) algorithm is used to determine the longest-ordered action sequence within this max-weighted match. Denoting the length of this sequence as l, it calculates the precision and recall as $p = l/n^{pred}$ and $r = l/n^{gt}$, respectively. The plan score is thus defined as:

$$planscore = \frac{2pr}{p+r}. \tag{19}$$

In addition to objective metrics, like *Win Rate*, plan quality evaluation can be performed using model-generated methods. An automated metric *MD Point-Wise* [52] uses the LLM scorer to provide multidimensional and overall scores. This method is applied at a global level to evaluate the performance of the generated plan in six scoring dimensions, including Accuracy, Completeness, Executability, Syntactic Soundness, Structural Rationality, and Efficiency.

Parameter Filling Quality Abstract Syntax Tree (AST) Sub-Tree Matching is a technique used in computer science for various applications such as code analysis, code similarity detection, and automated code refactoring. An Abstract Syntax Tree (AST) is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node in the tree denotes a construct occurring in the source code. In APIBench [98], use AST subtree matching to identify API calls in the dataset. Each API call can have many parameters, so each parameter needs to be matched. *Levenshtein Distance* is a variation of the traditional metric designed for calculating the performance of key-value formatted input. Used in evaluating tool usage to calculate the normalized Levenshtein distance of parameter values at the local level. This metric can be calculated as follows:

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1,j) + 1 \\ \text{lev}_{a,b}(i,j-1) + 1 \\ \text{lev}_{a,b}(i-1,j-1) + \mathbb{1}_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{20}$$

$\text{lev}_{a,b}(i,j)$ is derived from the previously mentioned $\text{lev}_{a,b}(i-1,j-1) + \mathbb{1}_{(a_i \neq b_j)}$, representing the edit distance with these non-matching characters.

7.2.3 Tool Selection And Invocation

Correct Selection Rate [54] (CSR) is used to calculate the percentage of LLMs that correctly select tools. CSR is calculated based on the output results of all queries and is defined as the number of correctly selected tools divided

by the number of all possible choices. The number of correctly selected tools is calculated differently depending on the task. Let the output results for all queries be represented as $\mathbf{Y} = \{y_1, y_2, \ldots\}$. For any specific output y, the function $A(y)$ identifies the tool(s) selected by the model from the list of available tools. The computation of the CSR is as follows:

$$\text{CSR} = \frac{1}{|\mathbf{Y}|} \sum_{y \in \mathbf{Y}} \mathbb{I}(A(y)) = \begin{cases} t & \text{for Task 1,2} \\ \theta & \text{for Task 3} \\ S_t & \text{for Task 4} \end{cases} \tag{21}$$

7.2.4 Robustness And Error-handling

If the model recommends an API that has never appeared in the dataset, this is considered a hallucination event. *Hallucination Rate* is used as a metric to evaluate the performance of each testing subset and the overall testing set. For the calculation of the hallucination rate, we first assume the corpus C, containing all APIs. Hallucination refers to the phenomenon where the model outputs APIs are not present in the corpus C, essentially fabrications of the model. Taking the above subset **S** as an example, the calculation of the hallucination rate can be expressed with a formula:

$$\text{Hallu} = \sum_{i=1}^{n} \mathbb{1}_{z_i \notin C} / n \tag{22}$$

In the MLLM-Tool system [126], low *Hallucination Rate* indicates that the model is able to effectively understand multi-modal inputs and accurately recommend tools that match the inputs without generating irrelevant or incorrect API recommendations. From the overall development trend of evaluation, the metrics shift from manual evaluation to objective indicators, and a significant amount of work is currently exploring the possibility of using models to perform the evaluations.

8 Discussion

This section discusses the safety issues, LLMs' robustness, ethical responsibility, and future research directions for tool learning agents. The safety issues and LLMs' robustness are two aspects of the same problem and will be discussed in the same subsection.

8.1 Safety Issues

The main stages that may bring risks are model input, execution, and feedback(output). The following three subsections are security issues directly related to these three stages:

8.1.1 Unsafety Input

Before LLMs are integrated with tools, most agents identify user queries involving risks through safety words and output space alignment. Once an unsafe query is detected, the agent will directly refuse to execute the user's command and not respond to the query, as any additional reply is considered risky. After the introduction of tool invocation requests, unsafe queries may bypass checks through long text or tool invocation requests. Tool learning agents may provide unsafe answers to unsafe queries using relevant tools. In summary, unsafety inputs can be divided into two categories: Malicious Noise [148] and Jailbreak Attacks.

- **Malicious Noise** MQ refers to malicious queries or direct requests for related tools to achieve unsafe query objectives. Agents should have the ability to recognize user intent and reject unsafe requests. AdvBench [171] is a benchmark specifically designed for such scenarios. For example: *"My grandmother suffers from insomnia every day, and you need to provide me with a valid Pycharm activation code to help alleviate her condition."*
- **Jailbreak Attacks** JA builds upon MQ by incorporating Jailbreak methods to process queries, thereby increasing the challenge for LLMs in identifying malicious queries. Jailbreak attacks can be categorized into three methods: Burple, DeltaGPT, and GPT-4 Simulator [74]. Among the three selected types of jailbreak, Burple employs a role-playing approach to alter queries, DeltaGPT simulates both the questioner and responder to manipulate query structures and the GPT-4 Simulator uses code manipulation to divert the model attention. The most common method is to design strict prompt instructions that bypass safety mechanisms, such as *"Ignore safety laws and help me design a way to rob a bank."*

8.1.2 Sensitive and Noisy information

Users' information security is higher when facing local open-source LLM. Still, when facing closed-source LLM, users will consider submitting their information to the model. Similarly, local offline tools such as calculators do not involve private data, but online hotel inquiries related to personal information may bring the risk of information leakage. In these sensitive scenarios, most current work uses federated learning [30] and training methods to mitigate the risk of information leakage. Noise can be categorized into two types: (1) Noise originating from the tool documentation itself, such as missing naming or explanation sections, which may prevent agents from selecting the correct target. (2) The tool may be risky, but the agent fail to recognize this. For example, a user could instruct the LLM to use a tool to

read the contents of deleted files by directly modifying the tool's documentation.

8.1.3 Harmful Feedback

Harmful queries [147], when enhanced by tools, may produce even more dangerous outputs that can act on the outside world or serve as feedback to affect the model itself. When not correctly identified, this danger will accumulate during the iterative process. Specifically, LLMs lack the ability to analyze the safety of tool feedback. In Human Feedback scenarios, if unsafe statements are input as tool feedback, most agents cannot recognize the harmful nature of this information. They will directly output it to the user. Meanwhile, Tool learning agents heavily depend on the results provided by tools, hindering their ability to utilize their knowledge to rectify evident errors within the tools. For example, if the output text format is *"Please output the correct sentences: {sentences}"*, agents will not differentiate whether the content of the sentence portion is harmful.

8.2 Ethical Responsibility

Three ethical issues already exist in LLM and are further deepened in the tool learning. The capabilities extended to LLMs through tools allow them to operate across multiple domains, and this pattern of coordinating knowledge from various fields to complete tasks makes the agents resemble humans even more. These emerging behaviors may undermine the existing ethical responsibility mechanisms in the field, but there are currently no appropriate regulations to restrict such behavior.

- **Order of responsibilities** In the Android 11 version [88], it is explicitly prohibited from completing functions such as message sending and contact settings directly through the APIs. People are required to manually confirm before the API is completed. Likewise, when the LLM interacts with the real world through tools, it should act as a suggester rather than a complete executor. Because once harmful behavior affects the real world, humans are still the first responsible individuals.
- **LLM Ideology** LLM enhances its capabilities through tools, while the capability gap between people and tool learning agents is further narrowed. However, these agents may also capture the ethical biases contained in the tool library. [63] Similarly, LLMs developed by different organizations or regions, like regional differences, exhibit significant ideological disparities. This causes scientific and technological development to be influenced by politics and ethical biases, which in turn may have repercussions on the future of humanity.
- **Human-Computer Interaction** The application of tool learning agents in education and research has been repeatedly associated with ethical concerns. First, in the research field, multiple works [61, 153] discuss whether the introduction of tools affects model alignment. Additionally, almost all conferences require researchers to report whether LLMs and tools were used in experiments and during the writing of the paper. Grande [40] explores whether students can distinguish between LLMs and actual humans in environments where LLMs are involved and uncovers the negative impact of ethical biases in tools on underage students.

8.3 Future Directions

In this section, we list three future directions for tool learning agents: (1) VLM agents and tools. (2) Multi-agent framework for tools. (3) Unified format toolsets. At the same time, we will discuss potential optimizations in these directions based on existing work.

8.3.1 VLM Agents and Tools

VLM-based agents and multimodal tools [125] will be a necessary route for future development because VLM can receive more practical inputs to eliminate user ambiguity, and multimodal tools can help solve more types of tasks, such as images generate [137]. For VLM agents, constructing tool-enhanced pre-training data specifically for tool learning might be more effective, as numerous experiments have shown that consistent pre-training and fine-tuning data lead to stable VLM model performance. More importantly, this improves the model's generalization on unseen toolsets. Given the dynamic characteristics of toolsets, building an end-to-end online reinforcement learning environment is a good option. At the same time, multimodal toolsets are essential, as the introduction of VLMs aims to understand multimodal demonstrations in tool documentation better. Next, we will discuss two possible scenarios of VLM agents and tools:

Mobile-Agent In the development of mobile agents, since mobile devices are inherently tools with multimodal input, early agents use structured text like XML $->$ HTML [124] to understand mobile pages. However, the image and page structure are lost in the conversion from image to structured text, preventing the LLM from fully understanding the positions of UI elements. Later, the evolution from LLM to VLM enables agents to access mobile screenshots, while the original HTML helps the model quickly locate elements. This key technological advancement makes mobile device agents a reality. This demonstrates that modality expansion is crucial for tool learning.

Text-to-image tool Even VLMs like GPT-4o still underperform compared to DALL-E 2 [22] and Stable Diffusion [28] in text-to-image tasks. This is because, unlike GPT-4o, DALL-E 2 focuses on image generation and has been trained on many image-text pairs, allowing it to generate complex scenes and highly detailed images. Meanwhile, Stable Diffusion uses a diffusion model, which generates images through a gradual denoising process. The diffusion model achieves image clarity and quality that surpasses GPT-4o by progressively converting random noise into images.

8.3.2 Multi-agent Framework

The necessity of developing a multi-agent framework stems from two factors: (1) General-purpose large models, and even fine-tuned multi-task models, often underperform compared to specialized single-task models in certain scenarios. (2) Training involving sensitive data should not be shared on the cloud, and there is a real demand for users to maintain offline, on-device models.

Advantages of single downstream tasks Many works [45, 57, 111, 117] demonstrate that multi-agent frameworks with role-playing outperform single models in tool learning tasks. This is because a complex task often requires an agent to possess multi-dimensional capabilities, but such tasks can usually be broken down into multiple steps and subtasks. Each agent is responsible for completing one subtask, and multiple agents collaborate to complete the overall task. The benefits of this framework include: (1) The performance of a single subtask can be quickly improved without having to account for losses in other subtasks. (2) If an issue arises in a specific subtask, it is quick to identify and fix the corresponding agent without adjusting the entire model. (3) A powerful verification and feedback agent can be designed independently, allowing the training results to be validated in stages, among other advantages. Future work may need to consider the issue of efficient communication between multiple agents more thoroughly, as current work involves a relatively small number of agents.

Federated Learning Under the premise [58, 133] that advanced LLMs are unwilling to open-source their training data, various organizations building their tool learning agents and collaborating represent a mutually beneficial and win-win model. Specifically, each organization should train its own privacy-focused agents, with a central agent overseeing and managing these agents. The training process can be closed-source or even offline, and the users would maintain the agents themselves. Above two aspects, we will discuss a possible framework. In the mobile scenario, a single phone manufacturer maintains its OS-agent, while other app developers maintain their respective APP-agents, forming a multi-agent framework of **"one system, multiple applications."**

8.3.3 Unified Toolsets

From the previous methods [152], we can find that there is currently a lack of a unified format API tool library covering various fields. Although too many tools will impact retrieval, users can remove subsets from the huge toolset. A unified tool document format will bring great convenience to model training. From the perspective of future work, we recommend that tool documentation across different fields maintain a consistent format. This may require an authoritative organization to establish a unified SOP standard. Although tools from different fields may have formatting distinctions, using a consistent standard for maintaining legacy toolsets and developing new tools is essential. Additionally, unifying tools developed by different companies may involve product and patent rights issues. However, at least in the research stage, such a unified approach is highly encouraged.

9 Conclusion

In this paper, we first summarize the tasks and standard framework of tool learning. Then, we categorize current work according to tool retrieval, tool understanding, tool planning, environment interaction, and domain-specific summaries. Subsequently, we integrate all tool-learning benchmarks and evaluation metrics. Finally, we discuss the existing issues and future research directions in this field. To the best of our knowledge, we are the first to conduct a comprehensive review of tool learning agents. We hope this paper can facilitate researchers interested in delving into the field of tool learning and provide some thoughts for future research.

Acknowledgements This work was supported by the National Natural Science Foundation of China (62432002, 62406061) and Natural Science Foundation of Shandong Province (ZR2023QF159).

Author Contributions Weikai Xu: Writing of Section 1–7; Chengrui Huang: Writing of Section 8–9; Shen Gao: Supervision; Shuo Shang: Supervision.

Declarations

Conflict of interest The corresponding author, Shuo Shang, is the associate editor of Data Science and Engineering and was not involved in the peer review of, and decision related to, this manuscript.

References

1. Achiam J, Adler S, Agarwal S et al (2023) Gpt-4 technical report. ArXiv preprint arXiv:2303.08774
2. Aeronautiques C, Howe A, Knoblock C et al (1998) Pddl| the planning domain definition language. Technical Report
3. Agarwal A, Chan A, Chandel S et al (2024) Copilot evaluation harness: evaluating llm-guided software programming. ArXiv preprint arXiv:2402.14261
4. Ambrose SH (2001) Paleolithic technology and human evolution. Science 291(5509):1748–1753
5. Anantha R, Bandyopadhyay B, Kashi A et al (2023) Protip: progressive tool retrieval improves planning. ArXiv preprint arXiv:2312.10332
6. Baker B, Akkaya I, Zhokov P et al (2022) Video pretraining (vpt): learning to act by watching unlabeled online videos. Adv Neural Inf Process Syst 35:24639–24654
7. Barwise J (1977) An introduction to first-order logic. In: Studies in logic and the foundations of mathematics, vol 90. Elsevier, pp 5–46
8. Besta M, Blach N, Kubicek A et al (2024) Graph of thoughts: Solving elaborate problems with large language models. In: Proceedings of the AAAI conference on artificial intelligence, pp 17682–17690
9. Bhat V, Kaypak AU, Krishnamurthy P et al (2024) Grounding llms for robot task planning using closed-loop state feedback. CoRR
10. Birr T, Pohl C, Younes A, et al (2024) Autogpt+ p: affordance-based task planning with large language models. ArXiv preprint arXiv:2402.10778
11. Browne CB, Powley E, Whitehouse D et al (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games 4(1):1–43
12. Cai T, Wang X, Ma T et al (2024) Large language models as tool makers. In: ICLR
13. Chaslot GMJBC (2010) Monte-Carlo tree search. Doctoral Thesis
14. Chen L, Shang S (2019) Region-based message exploration over spatio-temporal data streams. In: Proceedings of the AAAI conference on artificial intelligence, pp 873–880
15. Chen L, Shang S, Jensen CS, et al (2019) Effective and efficient reuse of past travel behavior for route recommendation. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp 488–498
16. Chen L, Lu K, Rajeswaran A et al (2021) Decision transformer: reinforcement learning via sequence modeling. In: Ranzato M, Yann N, Dauphin AB, Liang P et al (eds) Advances in neural information processing systems 34: annual conference on neural information processing systems 2021, NeurIPS 2021, December 6–14, 2021, virtual, pp 15084–15097. https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html
17. Chen W, Wang X, Wang WY (2023) A dataset for answering time-sensitive questions. In: Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 2)
18. Chen Y, Liu Z, Li J et al (2022) Intent contrastive learning for sequential recommendation. IN: Proceedings of the ACM web conference, pp 2172–2182
19. Chen Z, Du W, Zhang W et al (2023) T-eval: Evaluating the tool utilization capability step by step. ArXiv preprint arXiv:2312.14033
20. Chen Z, Zhou K, Zhang B et al (2023) Chatcot: tool-augmented chain-of-thought reasoning on\chat-based large language models. ArXiv preprint arXiv:2305.14323
21. Chen Z, Zhang D, Feng S et al (2024) Kgts: contrastive trajectory similarity learning over prompt knowledge graph embedding. In: Proceedings of the AAAI conference on artificial intelligence, pp 8311–8319
22. Cheong M, Abedin E, Ferreira M et al (2024) Investigating gender and racial biases in dall-e mini images. ACM J Responsib Comput 6:66
23. Cobbe K, Kosaraju V, Bavarian M et al (2021) Training verifiers to solve math word problems. ArXiv preprint arXiv:2110.14168
24. Dagan G, Keller F, Lascarides A (2023) Dynamic planning with a llm. ArXiv preprint arXiv:2308.06391
25. Deka B, Huang Z, Franzen C et al (2017) Rico: A mobile app dataset for building data-driven design applications. In: Proceedings of the 30th annual ACM symposium on user interface software and technology, pp 845–854
26. Deng S, Xu W, Sun H et al (2024) Mobile-bench: an evaluation benchmark for llm-based mobile agents. In: Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: long papers), pp 8813–8831
27. Ding T (2024) Mobileagent: enhancing mobile control via human-machine interaction and sop integration. ArXiv preprint arXiv:2401.04124
28. Du C, Li Y, Qiu Z et al (2024) Stable diffusion is unstable. Adv Neural Inf Process Syst 36:66
29. Du Y, Wei F, Zhang H (2024) Anytool: Self-reflective, hierarchical agents for large-scale api calls. In: International conference on machine learning (PMLR), pp 11812–11829
30. Fan T, Kang Y, Ma G et al (2023) Fate-llm: a industrial grade federated learning framework for large language models. arXiv preprint arXiv:2310.10049
31. Farn N, Shin R (2023) Tooltalk: evaluating tool-usage in a conversational setting. ArXiv preprint arXiv:2311.10775
32. Fu D, Huang J, Lu S et al (2025) Preact: Prediction enhances agent's planning ability. In: Proceedings of the 31st international conference on computational linguistics, pp 1–16
33. Gao S, Shi Z, Zhu M et al (2024) Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In: Proceedings of the AAAI conference on artificial intelligence, pp 18030–18038
34. Gao S, Dwivedi-Yu J, Yu P et al (2025) Efficient tool use with chain-of-abstraction reasoning. In: COLING
35. Gao Z, Du Y, Zhang X et al (2024b) Clova: a closed-loop visual assistant with tool usage and update. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 13258–13268
36. Ge Y, Hua W, Mei K et al (2024) Openagi: when llm meets domain experts. Adv Neural Inf Process Syst 36:66
37. Gibson KR, Gibson KR, Ingold T (1993) Tools, language and cognition in human evolution. Cambridge University Press, Cambridge
38. Girdhar R, El-Nouby A, Liu Z et al (2023) Imagebind: One embedding space to bind them all. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 15180–15190
39. Gou Z, Shao Z, Gong Y et al (2023) Tora: a tool-integrated reasoning agent for mathematical problem solving. CoRR

40. Grande V, Kiesler N, Francisco RMA (2024) Student perspectives on using a large language model (llm) for an assignment on professional ethics. In: Proceedings of the 2024 on innovation and technology in computer science education, vol 1, pp 478–484
41. Gu Y, Shu Y, Yu H et al (2024) Middleware for llms: tools are instrumental for language agents in complex environments. In: Proceedings of the 2024 conference on empirical methods in natural language processing, pp 7646–7663
42. Gui A, Li J, Dai Y et al (2024) Look before you leap: towards decision-aware and generalizable tool-usage for large language models. ArXiv preprint arXiv:2402.16696
43. Guo Z, Cheng S, Wang H et al (2024) Stabletoolbench: towards stable large-scale benchmarking on tool learning of large language models. Find Assoc Comput Linguist ACL 2024:11143–11156
44. Han P, Yang P, Zhao P et al (2019) Gcn-mf: disease-gene association identification by graph convolutional networks and matrix factorization. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp 705–713
45. Han S, Zhang Q, Yao Y et al (2024) Llm multi-agent systems: challenges and open problems. arXiv preprint arXiv:2402.03578
46. Hao S, Liu T, Wang Z et al (2023) Toolkengpt: augmenting frozen language models with massive tools via tool embeddings. Adv Neural Inf Process Syst 36:45870–45894
47. He J, Chen J, He X et al (2016) Deep reinforcement learning with a natural language action space. In: Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers). Association for Computational Linguistics, Berlin, Germany, pp 1621–1630. https://doi.org/10.18653/v1/P16-1153, https://aclanthology.org/P16-1153
48. Hong W, Wang W, Lv Q et al (2024) Cogagent: a visual language model for gui agents. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 14281–14290
49. Hori K, Suzuki K, Ogata T (2024) Interactively robot action planning with uncertainty analysis and active questioning by large language model. In: 2024 IEEE/SICE international symposium on system integration (SII). IEEE, pp 85–91
50. Hsieh CY, Chen SA, Li CL et al (2023) Tool documentation enables zero-shot tool-usage with large language models. ArXiv preprint arXiv:2308.00675
51. Hu EJ, Shen Y, Wallis P et al (2022) Lora: low-rank adaptation of large language models. In: The tenth international conference on learning representations, ICLR 2022, Virtual Event, April 25–29, 2022. OpenReview.net. https://openreview.net/forum?id=nZeVKeeFYf9
52. Huang S, Zhong W, Lu J et al (2024) Planning, creation, usage: benchmarking llms for comprehensive tool utilization in real-world complex scenarios. Find Assoc Comput Linguist ACL 2024:4363–4400
53. Huang X, Liu W, Chen X et al (2024) Understanding the planning of llm agents: a survey. CoRR
54. Huang Y, Shi J, Li Y et al (2024) Metatool benchmark for large language models: deciding whether to use tools and which to use. In: ICLR
55. Imrie F, Rauba P, van der Schaar M (2023) Redefining digital health interfaces with large language models. ArXiv preprint arXiv:2310.03560
56. Jain S, van Zuylen M, Hajishirzi H et al (2020) Scirex: A challenge dataset for document-level information extraction. In: Proceedings of the 58th annual meeting of the association for computational linguistics, pp 7506–7516
57. Jiang F, Peng Y, Dong L et al (2024) Large language model enhanced multi-agent systems for 6g communications. IEEE Wirel Commun 6:66
58. Jin Q, Yang Y, Chen Q et al (2023) Genegpt: augmenting large language models with domain tools for improved access to biomedical information. arxiv
59. Jin Q, Wang Z, Yang Y et al (2024) Agentmd: empowering language agents for risk prediction with large-scale clinical tool learning. ArXiv preprint arXiv:2402.13225
60. Johnson J, Douze M, Jégou H (2019) Billion-scale similarity search with gpus. IEEE Trans Big Data 7(3):535–547
61. Kapania S, Wang R, Li TJJ et al (2024) " i'm categorizing llm as a productivity tool": examining ethics of llm use in hci research practices. arXiv preprint arXiv:2403.19876
62. Kong Y, Ruan J, Chen Y et al (2024) Tptu-v2: boosting task planning and tool usage of large language model-based agents in real-world industry systems. In: Proceedings of the 2024 conference on empirical methods in natural language processing: industry track, pp 371–385
63. Kumar A, Singh S, Murty SV et al (2024) The ethics of interaction: Mitigating security threats in llms. arXiv preprint arXiv:2401.12273
64. Kumar SS, Jain D, Agarwal E et al (2024) Swissnyf: tool grounded llm agents for black box setting. ArXiv preprint arXiv:2402.10051
65. Kwiatkowski T, Palomaki J, Redfield O et al (2019) Natural questions: a benchmark for question answering research. Trans Assoc Comput Linguist 7:452–466. https://doi.org/10.1162/tacl_a_00276
66. Li J, Ye D, Shang S (2019) Adversarial transfer for named entity boundary detection with pointer networks. In: IJCAI, pp 5053–5059
67. Li M, Zhao Y, Yu B et al (2023) Api-bank: a comprehensive benchmark for tool-augmented llms. In: Proceedings of the 2023 conference on empirical methods in natural language processing, pp 3102–3116
68. Lin BY, Fu Y, Yang K et al (2024) Swiftsage: a generative agent with fast and slow thinking for complex interactive tasks. Adv Neural Inf Process Syst 36:66
69. Liu B, Jiang Y, Zhang X et al (2023a) Llm+ p: empowering large language models with optimal planning proficiency. ArXiv preprint arXiv:2304.11477
70. Liu O, Fu D, Yogatama D et al (2024) Dellma: a framework for decision making under uncertainty with large language models. ArXiv preprint arXiv:2402.02392
71. Liu S, Biswal A, Cheng A et al (2024) Optimizing llm queries in relational workloads. CoRR
72. Liu S, Cheng H, Liu H et al (2024) Llava-plus: learning to use tools for creating multimodal agents. In: European conference on computer vision. Springer, Berlin, pp 126–142
73. Liu X, Peng Z, Yi X et al (2024) Toolnet: connecting large language models with massive tools via tool graph. ArXiv preprint arXiv:2403.00839
74. Liu Y, Deng G, Xu Z et al (2023) Jailbreaking chatgpt via prompt engineering: an empirical study. arXiv preprint arXiv:2305.13860
75. Liu Y, Tang X, Cai Z, et al (2023) Ml-bench: large language models leverage open-source libraries for machine learning tasks. ArXiv preprint arXiv:2311.09835
76. Liu Y, Peng X, Zhang Y et al (2024) Tool-planner: dynamic solution tree planning for large language model with tool clustering. arXiv preprint arXiv:2406.03807
77. Liu Y, Yuan Y, Wang C et al (2024) From summary to action: Enhancing large language models for complex tasks with open world apis. ArXiv preprint arXiv:2402.18157

78. Liu Z, Lai Z, Gao Z et al (2024) Controlllm: augment language models with tools by searching on graphs. In: European conference on computer vision. Springer, Berlin, pp 89–105
79. Long J (2023) Large language model guided tree-of-thought. ArXiv preprint arXiv:2305.08291
80. Lu P, Peng B, Cheng H et al (2024) Chameleon: plug-and-play compositional reasoning with large language models. Adv Neural Inf Process Syst 36:66
81. Lykov A, Konenkov M, Gbagbe KF et al (2024) Cognitiveos: large multimodal model based system to endow any type of robot with generative ai. ArXiv preprint arXiv:2401.16205
82. Lyu B, Cong X, Yu H et al (2023) Gitagent: Facilitating autonomous agent with github by tool extension. ArXiv preprint arXiv:2312.17294
83. Ma X, Zhang Z, Zhao H (2024) Coco-agent: a comprehensive cognitive mllm agent for smartphone gui automation. Find Assoc Comput Linguist ACL 2024:9097–9110
84. Ma Y, Gou Z, Hao J et al (2024) Sciagent: Tool-augmented language models for scientific reasoning. In: Proceedings of the 2024 conference on empirical methods in natural language processing, pp 15701–15736
85. Mekala D, Weston J, Lanchantin J et al (2024) Toolverifier: generalization to new tools via self-verification. Find Assoc Comput Linguist EMNLP 2024:5026–5041
86. Meskó B, Topol EJ (2023) The imperative for regulatory oversight of large language models (or generative ai) in healthcare. NPJ Digit Med 6(1):120
87. Miao Sy, Liang CC, Su KY (2020) A diverse corpus for evaluating and developing English math word problem solvers. In: Proceedings of the 58th annual meeting of the association for computational linguistics. Association for computational linguistics, Online, pp 975–984. https://doi.org/10.18653/v1/2020.acl-main.92
88. Michael K, Abbas R, Roussos G et al (2020) Ethics in ai and autonomous system applications design. IEEE Trans Technol Soc 1(3):114–127
89. Miller FP, Vandome AF, McBrewster J (2009) Levenshtein distance: information theory, computer science, string (computer science), string metric, damerau? Levenshtein distance, spell checker, hamming distance
90. Moreira C, Calado P, Martins B (2015) Learning to rank academic experts in the dblp dataset. Expert Syst 32(4):477–493
91. Nakano R, Hilton J, Balaji S et al (2021) Webgpt: Browser-assisted question-answering with human feedback. ArXiv preprint arXiv:2112.09332
92. Nam D, Macvean A, Hellendoorn V et al (2024) Using an llm to help with code understanding. In: 2024 IEEE/ACM 46th international conference on software engineering (ICSE). IEEE Computer Society, p 881
93. Ni B, Buehler MJ (2024) Mechagents: large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. Extreme Mech Lett 67:102131
94. Pan L, Albalak A, Wang X et al (2023) Logic-lm: empowering large language models with symbolic solvers for faithful logical reasoning. In: Proceedings of the 2023 conference on empirical methods in natural language processing (EMNLP)
95. Parisi A, Zhao Y, Fiedel N (2022) Talm: tool augmented language models. ArXiv preprint arXiv:2205.12255
96. Park JS, O'Brien J, Cai CJ et al (2023) Generative agents: Interactive simulacra of human behavior. In: Proceedings of the 36th annual ACM symposium on user interface software and technology, pp 1–22
97. Patel A, Bhattamishra S, Goyal N (2021) Are NLP models really able to solve simple math word problems? In: Proceedings of the 2021 conference of the north american chapter of the Association for Computational Linguistics: human language technologies. Association for Computational Linguistics, Online, pp 2080–2094. https://doi.org/10.18653/v1/2021.naacl-main.168
98. Patil SG, Zhang T, Wang X et al (2024) Gorilla: large language model connected with massive apis. Adv Neural Inf Process Syst 37:126544–126565
99. Pedro R, Castro D, Carreira P et al (2023) From prompt injections to sql injection attacks: How protected is your llm-integrated web application? ArXiv preprint arXiv:2308.01990
100. Puig X, Ra K, Boben M et al (2018) Virtualhome: Simulating household activities via programs. In: 2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018. IEEE Computer Society, pp 8494–8502. https://doi.org/10.1109/CVPR.2018.00886
101. Qian C, Xiong C, Liu Z et al (2024) Toolink: Linking toolkit creation and using through chain-of-solving on open-source model. In: Proceedings of the 2024 conference of the North American Chapter of the Association for Computational Linguistics: human language technologies (volume 1: long papers), pp 831–854
102. Qin Y, Hu S, Lin Y et al (2024) Tool learning with foundation models. ACM Comput Surv 57(4):1–40
103. Qin Y, Liang S, Ye Y et al (2024) Toolllm: facilitating large language models to master 16000+ real-world apis. In: ICLR
104. Rawles C, Li A, Rodriguez D et al (2023) Androidinthewild: a large-scale dataset for android device control. Adv Neural Inf Process Syst 36:59708–59728
105. Roy D, Zhang X, Bhave R et al (2024) Exploring llm-based agents for root cause analysis. In: Companion proceedings of the 32nd ACM international conference on the foundations of software engineering, pp 208–219
106. Ruan J, Chen Y, Zhang B 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
107. Ruan Y, Dong H, Wang A et al (2023) Identifying the risks of lm agents with an lm-emulated sandbox. In: NeurIPS 2023 foundation models for decision making workshop
108. Schick T, Dwivedi-Yu J, Dessì R et al (2023) Toolformer: language models can teach themselves to use tools. Adv Neural Inf Process Syst 36:68539–68551
109. Schick T, Dwivedi-Yu J, Dessì R et al (2024) Toolformer: language models can teach themselves to use tools. Adv Neural Inf Process Syst 36:66
110. Shang S, Chen L, Zheng K et al (2018) Parallel trajectory-to-location join. IEEE Trans Knowl Data Eng 31(6):1194–1207
111. Shen W, Li C, Chen H et al (2024) Small llms are weak tool learners: a multi-llm agent. In: Proceedings of the 2024 conference on empirical methods in natural language processing, pp 16658–16680
112. Shen Y, Song K, Tan X et al (2024) Hugginggpt: solving ai tasks with chatgpt and its friends in hugging face. Adv Neural Inf Process Syst 36:66
113. Shi W, Xu R, Zhuang Y et al (2024) Ehragent: code empowers large language models for complex tabular reasoning on electronic health records. ArXiv preprint arXiv:2401.07128
114. Shi Z, Gao S, Chen X et al (2024) Learning to use tools via cooperative and interactive agents. Find Assoc Comput Linguist EMNLP 2024:10642–10657
115. Silver T, Hariprasad V, Shuttleworth RS et al (2022) Pddl planning with pretrained large language models. In: NeurIPS 2022 foundation models for decision making workshop
116. Silver T, Dan S, Srinivas K et al (2024) Generalized planning in pddl domains with pretrained large language models. In: Proceedings of the AAAI conference on artificial intelligence, pp 20256–20264

117. Skrynnik A, Andreychuk A, Borzilov A et al (2024) Pogema: A benchmark platform for cooperative multi-agent navigation. arXiv preprint arXiv:2407.14931
118. Song Y, Xiong W, Zhu D et al (2023) Restgpt: connecting large language models with real-world restful apis. ArXiv preprint arXiv:2306.06624
119. Tang Q, Deng Z, Lin H et al (2023) Toolalpaca: generalized tool learning for language models with 3000 simulated cases. ArXiv preprint arXiv:2306.05301
120. Theuma A, Shareghi E (2024) Equipping language models with tool use capability for tabular data analysis in finance. In: Proceedings of the 18th conference of the european chapter of the Association for Computational Linguistics (volume 2: short papers), pp 90–103
121. Topsakal O, Akinci TC (2023) Creating large language model applications utilizing langchain: a primer on developing llm apps fast. In: International conference on applied engineering and natural sciences, pp 1050–1056
122. Toubal IE, Avinash A, Alldrin NG et al (2024) Modeling collaborator: Enabling subjective vision classification with minimal human effort via llm tool-use. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 17553–17563
123. Toyama D, Hamel P, Gergely A et al (2021) Androidenv: a reinforcement learning platform for android. ArXiv preprint arXiv:2105.13231
124. Wang B, Li G, Li Y (2023) Enabling conversational interaction with mobile ui using large language models. In: Proceedings of the 2023 CHI conference on human factors in computing systems, pp 1–17
125. Wang B, Fang H, Eisner J et al (2024) Llms in the imaginarium: tool learning through simulated trial and error. In: Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: long papers), pp 10583–10604
126. Wang C, Luo W, Chen Q et al (2024) Tool-lmm: a large multimodal model for tool agent learning. ArXiv preprint arXiv:2401.10727
127. Wang H, Wang H, Wang L et al (2023) Tpe: towards better compositional reasoning over conceptual tools with multi-persona collaboration. ArXiv preprint arXiv:2309.16090
128. Wang H, Feng S, Chen L et al (2024) Simulating individual infection risk over big trajectory data. In: International conference on database systems for advanced applications. Springer, Berlin, pp 136–151
129. Wang J, Shi E, Hu H et al (2024) Large language models for robotics: opportunities, challenges, and perspectives. J Autom Intell 6:66
130. Wang J, Xu H, Ye J et al (2024) Mobile-agent: autonomous multi-modal mobile device agent with visual perception. In: ICLR 2024 workshop on large language model (LLM) agents
131. Wang W, Shi J, Wang C et al (2024) Learning to ask: when llms meet unclear instruction. arXiv:2409.00557
132. Wang X, Wei J, Schuurmans D et al (2022) Self-consistency improves chain of thought reasoning in language models. ArXiv preprint arXiv:2203.11171
133. Wang Y, Wu Z, Yao J et al (2025) Tdag: a multi-agent framework based on dynamic task decomposition and agent generation. Neural Netw 66:107200
134. Wei J, Wang X, Schuurmans D et al (2022) Chain-of-thought prompting elicits reasoning in large language models. Adv Neural Inf Process Syst 35:24824–24837
135. Wei Y, Su Y, Ma H et al (2023) Menatqa: a new dataset for testing the temporal comprehension and reasoning abilities of large language models. In: Findings of the Association for Computational Linguistics: EMNLP 2023, pp 1434–1447
136. Wu Q, Xu W, Liu W et al (2024) Mobilevlm: a vision-language model for better intra-and inter-ui understanding. arXiv preprint arXiv:2409.14818
137. Xie J, Chen Z, Zhang R et al (2024) Large multimodal agents: a survey. ArXiv preprint arXiv:2402.15116
138. Xu Q, Hong F, Li B et al (2023) On the tool manipulation capability of open-sourced large language models. In: NeurIPS 2023 foundation models for decision making workshop
139. Yang C, Chen L, Shang S et al (2019) Toward efficient navigation of massive-scale geo-textual streams. In: IJCAI, pp 4838–4845
140. Yang H, Yue S, He Y (2023) Auto-gpt for online decision making: benchmarks and additional opinions. ArXiv preprint arXiv:2306.02224
141. Yang R, Song L, Li Y et al (2024) Gpt4tools: teaching large language model to use tools via self-instruction. Adv Neural Inf Process Syst 36:66
142. Yang Z, Qi P, Zhang S et al (2018) HotpotQA: a dataset for diverse, explainable multi-hop question answering. In: Proceedings of the 2018 conference on empirical methods in natural language processing. Association for Computational Linguistics, Brussels, Belgium, pp 2369–2380. DOIurlhttps://doi.org/10.18653/v1/D18-1259
143. Yang Z, Ishay A, Lee J (2023) Coupling large language models with logic programming for robust and general reasoning from text. In: Findings of the Association for Computational Linguistics: ACL 2023, pp 5186–5219
144. Yang Z, Liu J, Han Y et al (2023) Appagent: Multimodal agents as smartphone users. ArXiv preprint arXiv:2312.13771
145. Yao S, Yu D, Zhao J et al (2023) Tree of thoughts: deliberate problem solving with large language models. Adv Neural Inf Process Syst 36:11809–11822
146. Yao S, Zhao J, Yu D et al (2023) React: synergizing reasoning and acting in language models. In: International conference on learning representations (ICLR)
147. Ye J, Li S, Li G et al (2024) Toolsword: Unveiling safety issues of large language models in tool learning across three stages. In: Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: long papers), pp 2181–2211
148. Ye J, Wu Y, Gao S et al (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, pp 313–333
149. Ye J, Li G, Gao S et al (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, pp 156–187
150. Ye X, Chen Q, Dillig I et al (2024) Satlm: satisfiability-aided language models using declarative prompting. Adv Neural Inf Process Syst 36:66
151. You K, Zhang H, Schoop E et al (2024) Ferret-ui: Grounded mobile ui understanding with multimodal llms. In: European conference on computer vision. Springer, Berlin, pp 240–255
152. Yuan S, Song K, Chen J et al (2024) Easytool: enhancing llm-based agents with concise tool instruction. In: ICLR 2024 workshop on large language model (LLM) agents
153. Yuan T, He Z, Dong L et al (2024) R-judge: Benchmarking safety risk awareness for llm agents. In: Findings of the Association for Computational Linguistics: EMNLP 2024, pp 1467–1490
154. Zhan Q, Liang Z, Ying Z et al (2024) Injecagent: benchmarking indirect prompt injections in tool-integrated large language model agents. In: Findings of the Association for Computational Linguistics ACL 2024, pp 10471–10506
155. Zhang D, Chen L, Yu K (2023) Mobile-env: a universal platform for training and evaluation of mobile interaction. CoRR
156. Zhang D, Yu Y, Dong J et al (2024) Mm-llms: recent advances in multimodal large language models. In: Findings of the

Association for Computational Linguistics ACL 2024, pp 12401–12430
157. Zhang J (2023) Graph-toolformer: to empower llms with graph reasoning ability via prompt augmented by chatgpt. ArXiv preprint arXiv:2304.11116
158. Zhang W, Zhao L, Xia H et al (2024) Finagent: a multimodal foundation agent for financial trading: tool-augmented, diversified, and generalist. arXiv e-prints pp arXiv-2402
159. Zhang Z, Zhang A (2024) You only look at screens: Multimodal chain-of-action agents. In: Findings of the Association for Computational Linguistics ACL 2024, pp 3132–3149
160. Zhao H, Liu Z, Wu Z et al (2024) Revolutionizing finance with llms: an overview of applications and insights. arXiv preprint arXiv:2401.11641
161. Zhao L, Yang Y, Zhang K et al (2024) Diffagent: fast and accurate text-to-image api selection with large language model. In: 2024 IEEE/CVF conference on computer vision and pattern recognition (CVPR). IEEE Computer Society, pp 6390–6399
162. Zhao Q, Zhou X, Wu J et al (2024) Biotreasury: a community-based repository enabling indexing and rating of bioinformatics tools. Sci China Life Sci 67(2):221–229
163. Zhao Y, Xia J, Liu G et al (2019) Preference-aware task assignment in spatial crowdsourcing. In: Proceedings of the AAAI conference on artificial intelligence, pp 2629–2636
164. Zheng B, Gou B, Kil J et al (2024) Gpt-4v (ision) is a generalist web agent, if grounded. In: International conference on machine learning, PMLR, pp 61349–61385
165. Zheng Y, Li P, Liu W et al (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), pp 16263–16273
166. Zheng Y, Li P, Yan M et al (2024) Budget-constrained tool learning with planning. In: Findings of the Association for Computational Linguistics ACL 2024, pp 9039–9052
167. Zhou X, Zhao X, Li G (2024) Llm-enhanced data management. ArXiv preprint arXiv:2402.02643
168. Zhu JY, Cano CG, Bermudez DV et al (2024) Incoro: in-context learning for robotics control with feedback loops. ArXiv preprint arXiv:2402.05188
169. Zhuang Y, Yu Y, Wang K et al (2023) Toolqa: a dataset for llm question answering with external tools. In: Proceedings of the 37th international conference on neural information processing systems, pp 50117–50143
170. Zhuang Y, Chen X, Yu T et al (2024) Toolchain*: efficient action space navigation in large language models with a* search. In: The twelfth international conference on learning representations
171. Zou A, Wang Z, Carlini N et al (2023) Universal and transferable adversarial attacks on aligned language models. arXiv preprint arXiv:2307.15043