
Tool Learning with Foundation Models

Yujia Qin¹, Shengding Hu¹, Yankai Lin^{2*}, Weize Chen¹, Ning Ding¹, Ganqu Cui¹,
Zheni Zeng¹, Xuanhe Zhou¹, Yufei Huang¹, Chaojun Xiao¹, Chi Han³, Yi Ren Fung³,
Yusheng Su¹, Huadong Wang¹, Cheng Qian¹, Runchu Tian¹, Kunlun Zhu⁸, Shihao Liang⁸,
Xingyu Shen¹, Bokai Xu¹, Zhen Zhang¹, Yining Ye¹, Bowen Li¹, Ziwei Tang⁵, Jing Yi¹,
Yuzhang Zhu¹, Zhenning Dai¹, Lan Yan¹, Xin Cong¹, Yaxi Lu¹, Weilin Zhao¹,
Yuxiang Huang¹, Junxi Yan¹, Xu Han¹, Xian Sun⁷, Dahai Li⁷, Jason Phang⁴,
Cheng Yang⁵, Tongshuang Wu⁶, Heng Ji³, Guoliang Li¹, Zhiyuan Liu^{1*}, Maosong Sun^{1*}

¹Tsinghua University, ²Renmin University of China, ³University of Illinois Urbana-Champaign,
⁴New York University, ⁵Beijing University of Posts and Telecommunications,
⁶Carnegie Mellon University, ⁷Zhihu Inc., ⁸ModelBest Inc.
qyj20@mails.tsinghua.edu.cn

Abstract

Humans possess an extraordinary ability to create and utilize tools, allowing them to overcome physical limitations and explore new frontiers. With the advent of recent powerful foundation models, artificial intelligence systems have the potential to be equally adept in tool use as humans. This paradigm, which is dubbed as *tool learning with foundation models*, combines the strengths of specialized tools and foundation models to achieve enhanced accuracy, efficiency, and automation in problem-solving. Despite its immense potential, there is still a lack of a comprehensive understanding of key challenges, opportunities, and future endeavors in this field. To this end, we present a systematic investigation and comprehensive review of tool learning in this paper. We first introduce the background of tool learning, including its cognitive origins, the paradigm shift of foundation models, and the complementary roles of tools and models. We recapitulate existing tool learning research and formulate a general tool learning framework: starting from understanding the user instruction, models should learn to decompose a complex task into several subtasks, dynamically adjust their plan through reasoning, and effectively conquer each sub-task by selecting appropriate tools. We also discuss how to train models for improved tool-use capabilities and facilitate the generalization in tool learning. Considering the lack of a systematic tool learning evaluation in prior works, we experiment with 18 representative tools and show the potential of current foundation models in skillfully utilizing tools. Finally, we discuss several open problems that require further investigation for tool learning, such as ensuring safe and trustworthy tool use, enabling tool creation with foundation models, and addressing personalization challenges. Overall, we hope this paper could inspire future research in integrating tools with foundation models. Relevant codes and datasets are publicly available for further research exploration¹.

“It is not only the violin that shapes the violinist, we are all shaped by the tools we train ourselves to use.”

— Edsger W. Dijkstra

*Corresponding authors.

¹<https://github.com/OpenBMB/BMTools>

Author contributions are listed in § 6.

Contents

1	Introduction	4
2	Background	6
2.1	Cognitive Origins of Tool Use	6
2.2	Tool Categorization: A User-Interface Perspective	7
2.3	Paradigm Shift of Foundation Models	8
2.4	Complementary Roles of Tools and Foundation Models	9
3	Tool Learning	10
3.1	Components of Tool Learning	10
3.1.1	Understanding the Components	10
3.1.2	Connecting the Components	11
3.2	The General Procedure: From Intent to Plan	12
3.2.1	Understanding Intent and Tools	12
3.2.2	Planning with Reasoning	14
3.3	Training Models for Improved Tool Learning	16
3.3.1	Learning from Demonstrations	17
3.3.2	Learning from Feedback	18
3.3.3	Generalizable Tool Learning	19
4	Application and Experiment	21
4.1	Evaluated Tools	21
4.2	Experiments	24
5	Discussion	25
5.1	Safe and Trustworthy Tool Learning	25
5.2	Tool Learning for Large Complex Systems	27
5.3	From Tool User to Tool Maker: AI’s Evolutionary Role	27
5.4	From General Intelligence to Personalized Intelligence	29
5.5	Tool Learning and Embodied Learning	30
5.6	Knowledge Conflicts in Tool Augmentation	31
5.7	Open Problems	33
6	Conclusion	34
	Contributions	34
A	Case Study	50
A.1	3D Models	50
A.2	Stock	52
A.3	Making Slides	53

CONTENTS

A.4	Movie Hunter	58
A.5	Search Engine	59
A.6	Chemicals Mining	60
A.7	Cooking Assistant	61
A.8	AI Painting	62
A.9	Navigating Knowledge Graphs	63
A.10	ALFWorld	64
A.11	Calculator	66
A.12	Weather	66
A.13	Online Shopping	67
A.14	Map	68
A.15	Processing Tables	69
A.16	Translation	71
A.17	Wikipedia	72
A.18	Database	73

1 Introduction

Tools are extensions of human capabilities designed to enhance productivity, efficiency, and problem-solving in human activities. Since the dawn of civilization, tools have been integral to the very essence of our existence (Washburn, 1960). Tool creation and utilization are motivated by a deep-rooted desire to overcome our physical limitations and discover new territories. More specifically, with advancements in tools, we can accomplish increasingly complex tasks with ease and efficiency, liberating time and resources to pursue more ambitious ventures. As such, tools have served as the crucial foundation upon which our cultural and social practices are built, transforming our modes of learning, communication, working, and entertainment, infusing these domains with new dimensions of accessibility and interactivity (Gibson et al., 1993). Throughout history, it is undeniable that human beings have played a pivotal role in the invention and manipulation of tools, which is a striking manifestation of intelligence (Shumaker et al., 2011). Given the rise of Artificial Intelligence (AI), one natural question is, does AI possess the potential to be equally adept and capable as its creators?

The prerequisite of the manipulation of tools is a thorough comprehension of the tools’ functionalities, as well as the ability to understand user intents and perform planning and reasoning for tool use. Before the advent of powerful foundation models (Bommasani et al., 2021), conducting tool-oriented AI research was exceedingly challenging. While certain basic tools could be fitted using shallow statistical models or deep neural models (Pomerleau, 1988; Mnih et al., 2013; Akkaya et al., 2019), their performance and stability remained inadequate to meet the demands of practical applications, let alone generalizing across various tools. This is due to the limitations of traditional supervised learning in capturing the complex operations essential for tool utilization and the insufficiency of trial-and-error approaches like reinforcement learning in mastering the extensive decision space associated with tool use. In a nutshell, the fundamental limitations in tool use by earlier AI lie in the insufficient capabilities of the models. Recently, the emergence of more capable foundation models, characterized by significantly improved capabilities, has rendered tool learning practicable. They have shown enormous semantic understanding capacity in diverse tasks, spanning the fields of natural language processing (NLP) (Brown et al., 2020), computer vision (CV) (Ramesh et al., 2022), biology (Jumper et al., 2021), etc. Additionally, they have demonstrated superior reasoning and decision-making abilities in complex interactive environments (Nakano et al., 2021). By harnessing the extensive world knowledge garnered during pre-training, they can perform grounded actions and interact with the real world. Notably, the emergence of ChatGPT (OpenAI, 2022) highlights the potential of foundation models to understand human intentions, automate intricate processes, and generate natural responses; the advent of GPT-4 (OpenAI, 2023) offers immense potential for multi-modal perception, which is essential to the real-world grounding ability.

Therefore, foundation models enable AI to harness tools, which can lead to more potent and streamlined solutions for real-world tasks. Foundation models are able to decipher complex data, simulate human-like planning capabilities, and generate a broad spectrum of outputs. Concurrently, specialized tools can be

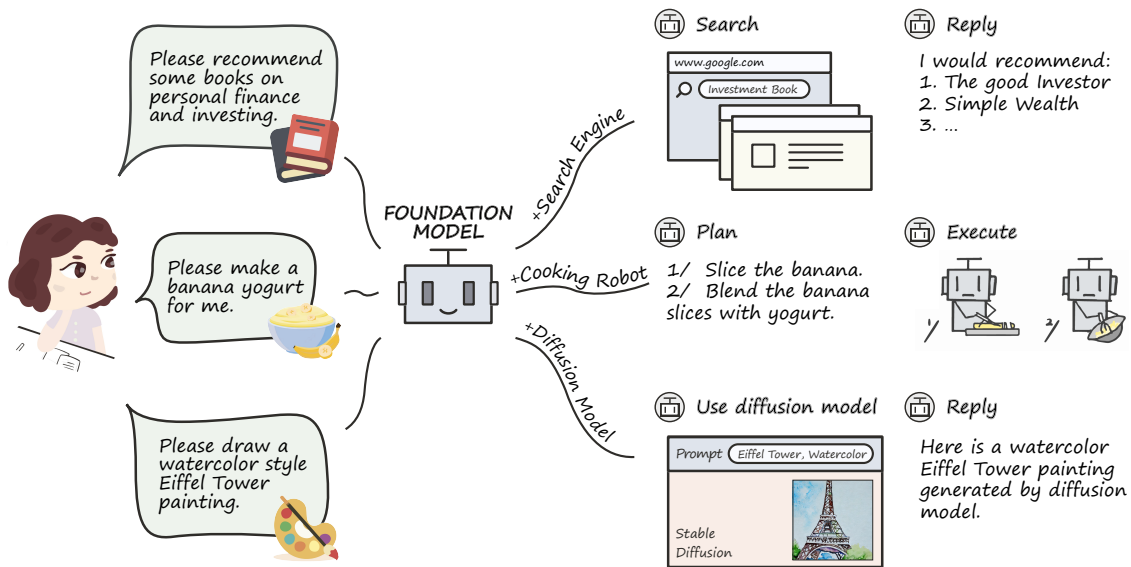


Figure 1: Tool learning paradigm aims to combine the strengths of specialized tools and foundation models.

employed to refine and target specific goals. The amalgamation of tools and models unveils vast potential where sophisticated procedures can be automated with limited human involvement. This paradigm, dubbed as **tool learning with foundation models** in this paper (Figure 1), aims to combine the strengths of specialized tools and foundation models, thereby culminating in greater accuracy, efficiency, and autonomy in problem-solving. Recent research has shed light on foundation models’ potential to exhibit a level of dexterity and finesse in tool use (Lazaridou et al., 2022; Nakano et al., 2021; Cobbe et al., 2021; Thoppilan et al., 2022; Huang et al., 2022b; Ahn et al., 2022; Yao et al., 2022a,b; Schick et al., 2023; Wu et al., 2023; Bubeck et al., 2023). Despite these breakthroughs, the efforts mainly focus on applying foundation models to specific tasks and domains with delicate algorithm designs. The current understanding of tool learning is still not comprehensive enough to estimate its characteristics and future developments. We believe that it is crucial to examine and summarize the current progress of tool learning with foundation models to explore their potential and challenges and to better pave the way for future technological advancements.

In this paper, we conduct a systematic investigation and comprehensive review of tool learning, attempting to build a full grasp of the key challenges, opportunities, and directions in this field. Before delving into the tool learning framework, we introduce essential **backgrounds** (§ 2), covering both tools and foundation models and their interaction. Specifically, we first recapitulate the cognitive origins of tool use in human history and its potential implications for tool use in AI systems (§ 2.1), followed by a categorization of tools from the perspective of the user interface (§ 2.2). Then we review the AI paradigm shift brought by foundation models and highlight the emergence and significance of tool learning (§ 2.3). After that, we discuss the complementary roles of tools and foundation models, and argue that integrating both can bring various advantages, such as improving interpretability, enhancing robustness, and delivering better user experiences (§ 2.4).

We present a comprehensive literature review for existing exploration in tool learning. While previous works often focus on specific aspects in isolation, we strive to formulate a general tool learning **framework** (§ 3.1), which comprises the controller (typically modeled using a foundation model), tool set, environment, perceiver, and human. Based on the unified framework, we review existing works of tool learning, highlight core research problems, and introduce their existing solutions as well as future explorations. The whole procedure (§ 3.2) of tool learning starts with a user instruction, and models are required to make an executable plan for tool execution. To bridge user instructions with appropriate tools, models should first learn to understand the user intents underlying the instruction (i.e., *intent understanding*) and understand the functionalities and usage of tools (i.e., *tool understanding*). Models should also learn to decompose a complex task into several subtasks, dynamically adjust their plan through reasoning, and effectively conquer each sub-task with the appropriate tools. Regarding the training strategy (§ 3.3) to facilitate models for improved tool utilization, we conclude with two mainstream methods: learning from demonstrations and learning from feedback. We discuss how to construct effective training supervision under different settings. To facilitate transferring the learned tool-use skills to new tools and situations, i.e., generalizable tool learning, it is important to design a unified interface that enables the model to interact with different tools in a standardized manner.

Considering the lack of a systematic tool learning evaluation in prior works, we conduct **experiments** (§ 4) on 18 representative tools based on our framework to investigate the efficacy and limitations of foundation models in tool manipulation. We demonstrate that state-of-the-art foundation models (e.g., ChatGPT) can effectively use tools to solve tasks with simple prompting. These results highlight the potential of using the foundation model as a general agent for tool learning.

Finally, we discuss the remaining important **research topics** (§ 5) for applying our general framework to real-world scenarios, including (1) **safety and trustworthiness**, where we emphasize the potential risks from adversaries, governance, and trustworthiness. We contend that careful considerations are necessary before deploying tool learning models in high-stakes scenarios (§ 5.1); (2) **tool learning for large complex systems**, where we showcase the unique characters of large complex systems and discuss the challenges in applying tool learning to these systems, such as complicated knowledge and function learning, representative data sampling with privacy concerns, and the strict requirements of efficient tool learning (§ 5.2); (3) **tool creation**, where we discuss the possibility that AI can also create new tools, challenging the long-held beliefs about what makes humans unique (§ 5.3); (4) **personalized tool learning**, where models provide tailored assistance to users in tool use. We highlight the challenges of aligning user preference with tool manipulation and introduce the shift from reactive to proactive systems, and the privacy-preserving concerns (§ 5.4); (5) **embodied learning**, where the intersection of tool learning and embodied agent enables digital embodiment and manipulation of embodied tools (§ 5.5); (6) **knowledge conflicts in tool augmentation**, where we review how tools can be leveraged to enhance models’ generation and the practical problems of knowledge conflicts, which can lead to inaccurate and unreliable model predictions (§ 5.6); (7) other **open problems**, such as viewing tool use capability as a measure for machine intelligence and tool learning for scientific discovery (§ 5.7). Overall, we

hope this paper could inspire further research in integrating tools with foundation models and developing more intelligent and capable AI systems.

2 Background

In this section, we first discuss the cognitive origins of human tool use (§ 2.1), followed by a tool categorization through the lens of the user interface (§ 2.2). Then we review the recent AI paradigm shift brought by foundation models (§ 2.3) and its significance in tool learning. After that, we examine the respective roles of specialized tools and foundation models in problem-solving, and discuss the benefits and challenges of their integration (§ 2.4).

2.1 Cognitive Origins of Tool Use

Tools have played a critical role in the long history of thousands of years of human evolution. Tools are commonly viewed as extensions of human beings, just as ancient fighting equipment and agricultural machinery were. Tool use is defined as a unique characteristic of human beings that is distinguished from other species (Von Eckardt, 1995). Throughout evolution, the ability to use tools has been essential for animals, particularly those with advanced intellectual development (Shumaker et al., 2011). For example, chimpanzees have been observed using stones or other materials to crack nuts (Boesch et al., 2019), while New Caledonian crows can craft and utilize two distinct types of hook tools to aid in capturing prey (Hunt, 1996). However, human tool behavior diverges from these observations in several ways. Humans can create much more complicated tools than other animals, such as converting our actions into fundamentally different mechanical actions in tools like hammers and pencils (Frey, 2007). Additionally, we can harness natural forces such as wind turbines to create tools (Shumaker et al., 2011). This ability may be attributed to our deep comprehension of cause-and-effect relations, which allows us to engage in technical reasoning (Osiurak & Reynaud, 2020).

Neural Basis of Tool Use. To better understand human tool use behaviors, researchers analyze the neural basis of tool observation and execution. It is verified that humans have parietal systems involved in grasping objects and using tools, and the anterior supramarginal gyrus activation of observing tool use is typical of human subjects, of which macaques do not exhibit (Orban & Caruana, 2014). This neurocognitive bases of tool observation may be related to the origins of cumulative technological evolution (e.g., the improvement in the efficiency and complexity of human tools and techniques over generations (Osiurak & Reynaud, 2020)) and some other human phenomena (Reynaud et al., 2019). While for the tool execution, researchers hold different views on manipulation-based versus reasoning-based approaches (Osiurak & Badets, 2016). The former claims that tool use has to be supported by the simulation of sensorimotor experiences, and the latter demonstrates the importance of reasoning based on mechanical knowledge in tool use. Nevertheless, the overall trend in cognitive science is understanding cognition as an enactive process that emphasizes interaction with the external world (Engel et al., 2013), and the feedback from observation, communication, and hands-on practice is important for mastering tool use.

Three Intelligence Levels of Tool Use. Besides, there are specific frameworks designed to discuss the level of intelligence represented by human tool use. For instance, “intoelligence” (Osiurak & Heinke, 2018) divides the tool use behavior into three modes: *assistive tool use* is usually passive and unaware (e.g., walking in the rain shelter corridor); *arbitrary tool use* requires active interaction (e.g., driving, using smart phones); *free tool use* further needs to comprehend and choose appropriate tools for the scenarios (e.g., cooking new dishes). In this framework, the three modes of tool use present a progressive relationship, and the authors assume that the key cognitive process for achieving free tool use is technical reasoning, which allows someone to learn new actions by observing others using, selecting, or making a tool instead of numerous practices.

Transition from Physical Tools to Conceptual Tools. Apart from tools in the physical world, we can also turn to more abstract tools. Take cognitive tools (Heyes, 2018) as an example: it refers to an auxiliary aid that facilitates higher-order thinking (e.g., multi-step critical analysis, the generation of creative problem-solving solutions). Cognitive tools can be classified based on the functionalities they provide (Lajoie & Derry, 2013). These include (1) supporting cognitive processes (e.g., documenting intermediate reasoning outcomes), (2) alleviating lower-level cognitive load to free up resources for advanced-level thinking, (3) enabling learners to engage in activities out of their reach and (4) allowing learners to generate and test hypotheses (e.g., simulated diagnoses for medical students).

Bridging the Gap between Human and Machine Tool Use. First, the abilities to manipulate tools are deeply rooted in our cognitive and perceptual systems and have evolved over millions of years. In contrast, foundation models rely primarily on statistical patterns of pre-training data, and significant gaps still exist between the tool-use capabilities of foundation models and their human counterparts. Humans can perceive the properties of tools, understand their functionalities, and identify the appropriate tools for each task. Gaining insights from this, we investigate and discuss how foundation models can learn such a process in § 3.2.1. Second, humans excel at breaking down complex tasks into smaller, manageable subtasks and deftly manipulating tools to accomplish each sub-task. However, foundation models lack the physical embodiment and sensory experience necessary to fully understand and utilize tools. As a result, these models often struggle with tasks that require higher-order reasoning and adaptivity, and they cannot trustfully integrate multiple sources of knowledge and tools effectively. We will discuss how to better make executable plans leveraging models' reasoning abilities in § 3.2.2. Furthermore, current algorithms for adapting foundation models to learn specific tools generally require huge amounts of supervised data (Nakano et al., 2021; Reed et al., 2022), hindering their generalization and transferability to broader types of tools or novel situations. Hence we first summarize the training strategies for tool learning (§ 3.3.1 and § 3.3.2) and discuss how to facilitate the generalization and transferability of tool learning (§ 3.3.3).

2.2 Tool Categorization: A User-Interface Perspective

The growing number and complexity of tools in our world make it increasingly important to understand and group them in a meaningful way. A system for classifying these tools helps us better grasp their uses, benefits, and potential for growth. In this paper, our focus is particularly on those tools that can be manipulated using instructions in conjunction with foundation models. We introduce a taxonomy that sorts these tools based on their modes of expression and interaction. As depicted in Figure 2, this taxonomy incorporates three levels of interaction, arranged from the most tangible to the least. *The physical level* involves direct physical interactions with tools. *The graphical user interface (GUI) level* facilitates user interaction with visual representations of tools. *The program level* involves users engaging directly with the underlying source code of tools.

Physical Interaction-based Tools. We start with the most tangible genre of tools, physical interaction-based tools. As the name suggests, this class of tools involves direct interactions with the physical world, including devices like robots, sensors, and wearables that could physically impact the environment. Physical interaction tools have the capability to sense and respond to the physical environment of users, making them useful in a wide range of applications, from manufacturing to healthcare and education. Physical interaction tools are close to the real world, and they have the potential to substantially improve efficiency and productivity. For example, robots can perform from simple to intricate, even adventurous tasks to reduce human errors and labor costs. Sensors can collect valuable data, such as temperature and pressure, allowing for real-time monitoring and optimization of industrial processes. Wearables, on the other hand, provide users with a personalized experience by tracking physiological or environmental parameters, promoting health and safety. It is worth noting that although the output of such tools interacts with the real world at the physical level, users may also create the input of the tools at the GUI or source code level.

GUI-based Tools. Some tools allow users to manipulate them through an interactive interface, i.e., visual representations of tools, with pre-defined operations. These tools, defined as GUI-based tools, do not have a direct impact on the physical world. The GUI interface typically includes buttons, menus, text boxes, and other graphical elements that allow users to interact with the underlying system. These tools are extensively employed in various industries and applications such as software development, data analysis, and design. Particularly, GUI-based tools can improve productivity by streamlining workflows and automating repetitive tasks. GUI-based tools could considerably simplify complex tasks and reduce the learning curve for non-technical users. From this viewpoint, tool learning with foundation models share the same primary goal, which simplifies intricate tasks to a natural language format. Representative GUI-based tools are usually well-developed software such as browsers, Microsoft Office, Adobe PhotoShop, etc. These applications showcase the versatility that graphical interfaces can provide and enable users to access and manipulate complex features within the software. On the other hand, the main limitation of GUI-based tools is that they may not provide the flexibility and customizability of command-line interfaces or APIs. In specific scenarios that require rapid and mass responses, as well as greater and flexible control, the visual interface may not always be the most effective way to interact with a system.

Program-based Tools. The innermost layer of tools that users can access is the source code, offering a high degree of flexibility for the input and output of these program-based tools. Program-based tools are software tools primarily designed for use through programming interfaces rather than visual interfaces. They can take

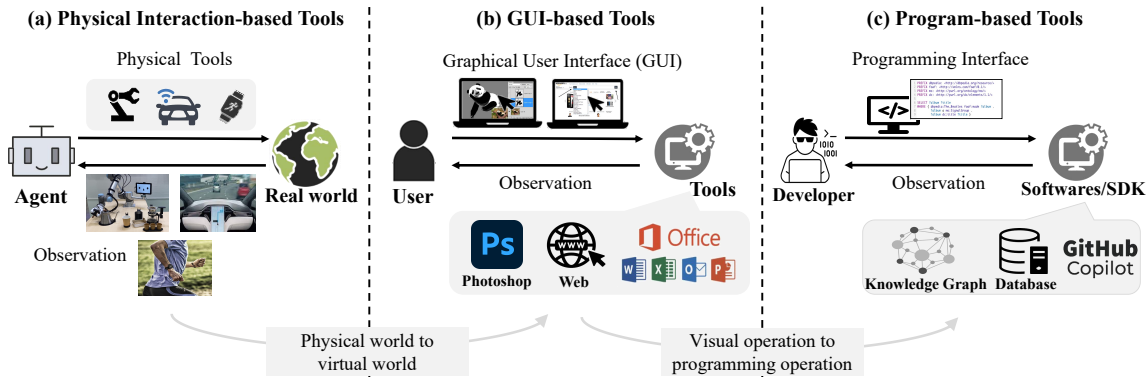


Figure 2: Tool categorization from the perspective of the user interface: (1) physical interaction-based tools, (b) GUI-based tools, and (c) program-based tools.

various forms, including declarative languages, programming libraries, software development kits (SDKs), and even neural network-based tools. These tools are typically used by developers or technical users who possess a deeper understanding of the underlying data, system or technology, with which the users could complete complex software applications. The main advantage of program-based tools is that they provide greater flexibility and customizability than GUI-based tools, and users can build more sophisticated solutions for current problems. As a result, such tools also have a steeper learning curve than GUI-based tools, they require a greater degree of technical expertise and programming knowledge, which may not be accessible to non-technical users. For example, program-based tools can be more time-consuming to set up and configure and may require more maintenance and support in the learning process. It is noteworthy that, although these tools pose difficulties for human beings in terms of the learning curve, they may not have the same level of challenges for foundation models.

It can be seen that the above three interaction modes have varying levels of connectivity with the tool kernel. They are not strictly mutually exclusive but indicate a tendency to intermingle with each other. Human beings have the ability to deal with complex tasks by flexibly executing tools of different types. In this paper, we contend that regardless of the tool type, it is fundamentally possible to leverage foundation models to execute them by setting up intermediary interfaces. We will introduce ways to unify the interface of different tools for foundation models in § 3.3.3.

2.3 Paradigm Shift of Foundation Models

In recent years, the field of natural language processing (NLP) has undergone a paradigm shift, marked by the advent of pre-trained language models (PLMs) (Devlin et al., 2019; Bommasani et al., 2021; Han et al., 2021). Prior to this breakthrough, NLP was a challenging field that necessitated designing separate learning objectives for distinct research domains, such as dependency parsing (Kübler et al., 2009), named entity recognition (Nadeau & Sekine, 2007), and summarization (Nenkova & McKeown, 2012). Despite the successful design of effective models and methods for these specific tasks, the separated nature of this paradigm impeded progress toward a holistic comprehension of language, thereby limiting its potential.

The invention of PLMs changes this paradigm. Building on Transformers (Vaswani et al., 2017), PLMs are trained on massive corpora, from which general linguistic ability and world knowledge are learned. This technique has expedited the unification of NLP tasks, giving rise to the *pre-train-then-fine-tune* paradigm, which has achieved new state-of-the-art performance on several NLP benchmarks, such as GLUE (Wang et al., 2019b) and SuperGLUE (Wang et al., 2019a). At this stage, each task shares the same starting point and only diverges as the task-specific adaptation proceeds. The fusion of task paradigms is still ongoing. T5 (Raffel et al., 2020) transforms all NLP tasks into a text-to-text format with textual descriptions, while GPT-3 (Brown et al., 2020) has discovered that introducing appropriate textual prompts can yield the desired output for specific tasks. Prompts, essentially serving as a natural language interface, are widely believed to stimulate the knowledge learned by PLMs during pre-training. Prompts can enable downstream tasks to be executed without updating model parameters for big models such as GPT-3. Research even suggests that with appropriate prompt guidance, models can perform complex reasoning tasks (Wei et al., 2022c; Wang et al., 2022a). Also, prompts formulated in a natural language format possess remarkable generalization capabilities. Specifically,

models that have undergone fine-tuning using diverse instructions are able to effectively generalize to new, unseen data (Wei et al., 2022a; Sanh et al., 2022). Overall, prompts demonstrate a proof-of-concept that uses PLMs as the underlying infrastructure and natural language as the medium to uniformly perform various tasks. A highly successful example is ChatGPT, where all the natural language understanding and generation processes are accomplished through conversational interactions.

Nevertheless, there exist numerous tasks that transcend the scope of purely natural language. For instance, generating presentation slides², constructing 3D models via CAD applications, and scheduling meetings through the analysis of team member calendars are examples of complex tasks that have not been defined in traditional artificial intelligence. Fortunately, the strong generalization ability of PLM enables us to use natural language as a medium to accomplish these tasks by manipulating tools (Zeng et al., 2022). Essentially, the key to tool learning is to decompose complex tasks into sub-actions, tokenize actions in the form of natural language and convert them into executable instructions that can be understood by specific tools. Language models serve as “translators”, making complex tasks more accessible to individuals without specialized technical knowledge. The potential applications of tool learning are vast and exciting, ranging from automated customer service and personal assistants to self-driving cars and even space exploration. By enabling machines to understand and interact with human language in a more natural and nuanced way, we can unlock new possibilities for collaboration and problem-solving that were previously impossible. We anticipate that tool learning will prove instrumental in facilitating the integration of diverse tasks through shared tooling. Thus, while natural language interfaces have enabled unification within the realm of language (Hao et al., 2022), the challenges posed by non-linguistic tasks necessitate a more advanced approach to leveraging both natural language and tool learning. By harnessing the power of natural language, we can create systems that are capable of understanding and adapting to the complex and dynamic world around us, opening up new avenues for innovation and discovery.

2.4 Complementary Roles of Tools and Foundation Models

The integration of specialized tools and foundation models represents a promising approach for harnessing the unique strengths of both. By incorporating foundation models’ understanding and reasoning capabilities into specialized tools, we can create intelligent tools capable of performing more complex tasks than either specialized tools or foundation models alone. Specifically, the amalgamation of both confers a multitude of benefits as follows.

Benefits of Tools. Tools that are designed to streamline concrete and specific objectives bring several benefits for tool learning: (1) **Mitigation for Memorization.** Although foundation models have demonstrated an exceptional ability to memorize (Carlini et al., 2021, 2022, 2023), they are not capable of memorizing every piece of training data. Furthermore, foundation models are often prompted with a relatively short context during model generation, thus not all the memorized knowledge can be properly steered (Mialon et al., 2023). Additionally, memorization alone does not support the real-time coverage of up-to-date knowledge, especially in light of the potentially infinite possibilities of novel requests from users. Besides, foundation models are also criticized to hallucinate knowledge (Roller et al., 2021; Shuster et al., 2021) by generating seemingly plausible but non-factual content. This is unacceptable in applications like financial transactions that require the results are 100% correct. Given the above factors, it is necessary to augment foundation models with real-time tool execution to mitigate limitations in memorization. For instance, a significant proportion of the memorization burden can be offloaded to the search engine and database systems (of different modes) if foundation models can learn how to utilize it. (2) **Enhanced Expertise.** Specialized tools are designed to cater to specific domains with functionalities that are not available in foundation models. As a result, they are better suited to address the needs of domain-specific tasks, such as Wolfram³ for scientific calculation, through the utilization of tailored algorithms. Instead of solely relying on the foundation model to accomplish the task, models could invoke appropriate tools to generalize to a wider range of tasks that are beyond their capabilities. (3) **Better Interpretability.** Foundation models are criticized for lacking transparency in their decision-making process (Linardatos et al., 2020), which can be a significant concern in applications such as healthcare or finance, where interpretability is critical for making informed decisions. In contrast, the process of tool execution reflects the whole process of how models solve complex requests, which allows for better interpretability and transparency. Users can easily understand why certain tools are called and how they contribute to the final output, which can improve trust and facilitate human-machine collaboration. (4) **Improved Robustness.** Foundation models are susceptible to adversarial attacks (Wallace et al., 2019; Jin

²<https://www.microsoft.com/en-us/microsoft-365>

³<https://www.wolframalpha.com/>

et al., 2020), where slight modifications to the input can flip the model prediction. This is because these models heavily rely on statistical patterns in the training data. Conversely, tools are designed specifically for their intended use cases, which may be agnostic to the input perturbation. This makes tools more resistant to adversarial attacks. Overall, incorporating tools into the workflow of foundation models can improve the robustness of the system and reduce the risk of malicious attacks. This harmonious interplay between tools and models can enhance the reliability of the system against unpredictable real-world environments. In § 4 and appendix A, we use concrete examples to show how tools can enhance the model’s capabilities in various tasks.

Benefits of Foundation Models. Foundation models can provide a solid basis for understanding, planning, reasoning, and generation, which bring several benefits for tool learning as follows: (1) **Improved Decision-Making and Reasoning Abilities.** Foundation models are trained on vast amounts of data, enabling them to acquire world knowledge across a wide range of domains. If properly steered, such knowledge can be wielded to perform decision-making and planning over prolonged time horizons (Huang et al., 2022a). Besides, foundation models have demonstrated remarkable reasoning abilities (Wei et al., 2022c; Wang et al., 2022a), thereby enabling them to extrapolate the consequences of actions and make judicious decisions. These reasoning abilities are particularly useful for tasks requiring a deep understanding of cause-and-effect relations (§ 3.2.2). (2) **Better User Experience.** Benefitting from the powerful intent understanding capability of foundation models, tool learning could revolutionize the way we interact with machines and liberate users from the cognition load, allowing them to engage in higher-order thinking and decision-making processes. This, in turn, fosters a seamless and more natural language-based interaction paradigm that revolutionizes traditional graphical user interfaces (GUIs). The user only needs to provide high-level guidance and direction, and the model will seamlessly comprehend the user’s intent, thereby delivering more personalized and precise responses. In addition, tool learning has the potential to democratize access to complex tools. With the aid of foundation models, even novice users can easily and quickly get started with a new tool, regardless of their prior experience or technical expertise. This not only reduces the barriers to entry for new users but also unlocks a wealth of possibilities for innovation and creativity. However, it should be noted that human-model collaboration in tool use also triggers ethical concerns, which will be discussed in § 5.7.

3 Tool Learning

In this section, to unify existing efforts and promote a comprehensive understanding of tool learning, we first present a general framework, which encompasses four fundamental components, namely tool set, environment, controller, and perceiver, as detailed in § 3.1. Subsequently, we provide an elaborate discussion of the general procedure of tool learning in § 3.2. Lastly, we delve into the training methods for tool learning and discuss how to achieve generalizable tool learning in § 3.3.

3.1 Components of Tool Learning

How can we enable foundation models to leverage the strengths of specialized tools to accomplish complex tasks? To better answer this question, we frame tool learning with four components as shown in Figure 3. Each component has its own characteristics and functions (§ 3.1.1), but they also interact with each other closely (§ 3.1.2).

3.1.1 Understanding the Components

We first introduce each component and explain how they contribute to the tool learning process.

Tool Set. Serving as the fundamental ingredient of tool learning, the tool set $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \dots\}$ contains a collection of different tools that have different functionalities. As we have elaborated in § 2.2, a tool in \mathcal{T} can have different interfaces. In the following sections, we mainly take Application Programming Interface (API) as the example to illustrate how to interact with tools. Here we define an API as any function that can take the output of the foundation model as its input. For instance, for a weather API, the input to the API may be a location and time, and the output may contain the temperature or wind speed.

Environment. The environment \mathcal{E} is the world where the tools operate, which provides the *perceiver* with the execution results of tools. It provides the infrastructure necessary for tool execution, which can be either virtual or real. The former refers to a simulated environment that allows the model to interact with a digital

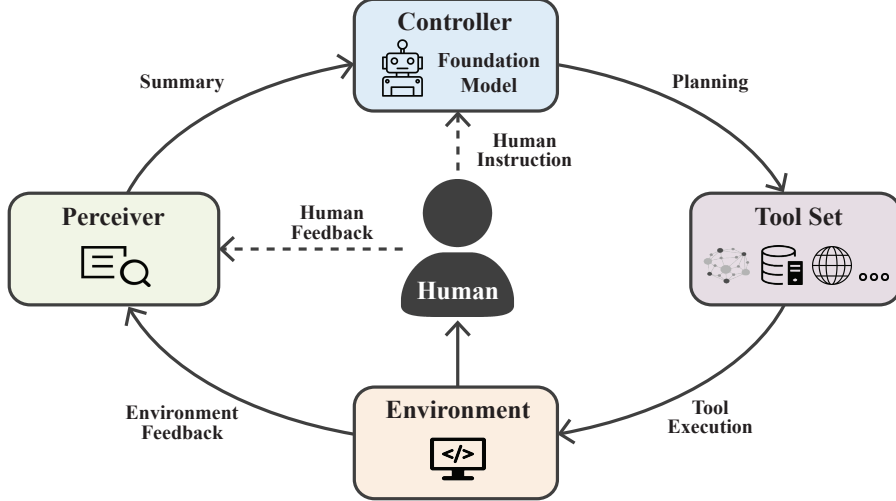


Figure 3: Illustration of the tool learning framework, where we display the human user and four core ingredients of the framework: tool set, controller, perceiver, and environment. The user sends an instruction to the controller, which then makes decisions and executes tools in the environment. The perceiver receives feedback from both the environment and the user and summarizes them to the controller.

representation of the tool, while a real environment involves actual interaction with the physical tool. Virtual environments have the advantage of being easily accessible and replicable, allowing for more cost-effective training for models. However, virtual environments may not fully replicate the complexities of the real-world environment, leading to overfitting and poor generalization (Hansen et al., 2021). On the other hand, real environments provide a more realistic context but may be more challenging to access and involve greater costs.

Controller. The controller \mathcal{C} serves as the “brain” for tool learning framework and is typically modeled using a foundation model. The purpose of the controller \mathcal{C} is to provide a feasible and precise plan for using tools to fulfill the user’s request. To this end, \mathcal{C} should understand user intent as well as the relationship between the intent and available tools, and then develop a plan to select the appropriate tools for tackling tasks, which will be discussed in § 3.2.1. In cases where the query is complex and targets a high-level task, \mathcal{C} may need to decompose the task into multiple sub-tasks, which requires foundational models to have powerful planning and reasoning capabilities (§ 3.2.2).

Perceiver. The perceiver \mathcal{P} is responsible for processing the user’s and the environment’s feedback and generating a summary for the controller. Simple forms of feedback processing include concatenating the user and environment feedback or formatting the feedback using a pre-defined template. The summarized feedback is then passed to the controller to assist its decision-making. By observing this feedback, the controller can determine whether the generated plan is effective and whether there are anomalies during the execution that need to be addressed. Under more complex scenarios, the perceiver should be able to support multiple modalities, such as text, vision, and audio, to capture the diverse nature of feedback from the user and the environment.

3.1.2 Connecting the Components

Formally, assume we have a tool set \mathcal{T} , which the controller can utilize to accomplish certain tasks. At time step t , environment \mathcal{E} provides feedback e_t on the tool execution. The perceiver \mathcal{P} receives the user feedback f_t and the environment feedback e_t , and generates summarized feedback x_t . Typically, the perceiver can be achieved by pre-defined rules (e.g., concatenating f_t and e_t) to form x_t , or modeled with complex neural models. The controller \mathcal{C} generates a plan a_t , which selects and executes an appropriate tool from \mathcal{T} . This process can be formulated as the following probability distribution:

$$p_{\mathcal{C}}(a_t) = p_{\theta_{\mathcal{C}}}(a_t \mid x_t, \mathcal{H}_t, q), \quad (1)$$

where $\theta_{\mathcal{C}}$ denotes the parameters of \mathcal{C} , q denotes the user query or instruction, and $\mathcal{H}_t = \{(x_s, a_s)\}_{s=0}^{t-1}$ denotes the history feedback and plans. In its simplest form, a generated plan a_t can simply be a specific action for tool

execution. \mathcal{C} can also synergize its reasoning process with the action prediction, where a_t may additionally contain the reasoning traces that explain which sub-task should be solved next and which tool to choose for solving the sub-task. It is worth noting that if the dependence on x_s is removed from Equation (1), the resulting probability distribution becomes equivalent to autoregressive language modeling. From this perspective, the controller additionally grounds the foundation model to the environment and the tool set. Moreover, we can factorize Equation (1) as follows:

$$p_{\theta_c}(a_t | x_t, \mathcal{H}_t, q) = \sum_{\mathcal{T}_i \in \mathcal{T}} p_{\theta_c}(a_t | \mathcal{T}_i, x_t, \mathcal{H}_t, q) \times p_{\theta_c}(\mathcal{T}_i | x_t, \mathcal{H}_t, q), \quad (2)$$

The decomposition reveals that the construction of the plan a_t involves two subtasks: selecting the appropriate tool based on the user intent and deciding the actions to execute using the selected tool. For instance, given an instruction such as “I want to book a flight to Beijing next week”, the controller \mathcal{C} first infers that the user’s goal is to reserve a flight, with Beijing as the destination and the next week as the travel time. The model then selects the airline reservation system as the tool. Finally, it inputs the time and destination as the preliminary plan. In the process of making a reservation, we may face unexpected situations such as the unavailability of flights to Beijing in the next week. To cope with these anomalies, we can further equip \mathcal{C} with the ability to reason about the current context and generate alternative plans, as we will discuss in detail in § 3.2.2.

After a plan a_t is generated, it will be executed in \mathcal{E} , and the resulting feedback e_{t+1} from \mathcal{E} will be passed on to the perceiver. The above process repeats for multiple rounds until the controller accomplishes the task. The overall objective is to find an action sequence $\{a_t\}$ that ultimately fulfills the task specified by the user instruction q . Note after tool execution, the controller may additionally integrate the execution results into a plausible response for the user (see details in § 5.6).

3.2 The General Procedure: From Intent to Plan

As formulated in § 3.1.2, the general procedure of tool learning necessitates intricate interplay among different components. In this section, we will further elaborate on the key issues involved in this procedure.

3.2.1 Understanding Intent and Tools

To accurately fulfill the task specified by the user query q , the controller needs to understand two aspects: (1) the underlying intent of the user, which involves recognizing and formalizing the natural language q as a high-level task (i.e., *intent understanding*); (2) the tool set \mathcal{T} , which entails comprehending the functionality and objective of each tool within it (i.e., *tool understanding*). By understanding both aspects, the controller can bridge the gap between the user’s intent and the tool set, which is the pre-requisite for connecting controller \mathcal{C} , the user, and tool set \mathcal{T} in Figure 3.

Intent Understanding. Understanding user intent is a long-standing research topic in NLP (Jansen et al., 2007; Sukthankar et al., 2014), which involves comprehending the underlying purpose of a user query. Intent understanding is essential in scenarios requiring human-computer interaction, such as developing advanced conversational agents capable of conducting intricate and nuanced dialogues with users. It requires learning a mapping from the instruction space to the model’s cognition space. By accurately identifying the user intent, the controller can provide more personalized responses with a better user experience.

Recent explorations in instruction tuning (Wei et al., 2022a) demonstrate that foundation models can possess extraordinary proficiency in comprehending user instructions. Prior work has shown that fine-tuning large language models on a collection of datasets templated with human instructions allows models to generalize even to instructions for unseen tasks (Wei et al., 2022a; Mishra et al., 2022; Sanh et al., 2022; Bach et al., 2022; Ouyang et al., 2022). Promisingly, such generalization ability can further be enhanced by scaling up both the model size and the quantity or diversity of training instructions (Iyer et al., 2022). Despite the impressive intent understanding capabilities, challenges still exist in real-world tool learning scenarios: (1) **Understanding Vague Instructions.** The first challenge is dealing with the inherent vagueness and ambiguity in the user query. Many user queries are inherently imprecise and can even be polysemous, requiring the controller to rely on contextual cues and background knowledge to infer the user’s intended meaning. One possible solution is to actively interact with users to clarify any ambiguity, such as asking for clarifications about a previous user query. (2) **Generalization to Diverse Instructions.** Another challenge is having the models generalize to more diverse user instructions. As the intent space is theoretically infinite, it is almost impractical for foundation models to be exposed to every real-world intention during training. In addition, the challenge of personalization arises from the fact that each individual has their own unique way of expressing intentions,

Zero-shot Prompting: Here we provide a tool (API) "forecast_weather(city:str, N:int)", which could forecast the weather about a city on a specific date (after N days from today). The returned information covers "temperature", "wind", and "precipitation".
Please write codes using this tool to answer the following question: "What's the average temperature in Beijing next week?"

Few-shot Prompting: We provide some examples for using a tool. Here is a tool for you to answer question:

Question: "What's the temperature in Shanghai tomorrow?"

```
return forecast_weather("Shanghai", 1) ["temperature"]
```

Question: "Will it rain in London in next two days?"

```
for i in range(2):
    if forecast_weather("London", i+1) ["precipitation"] > 0:
        return True
return False
```

Question: "What's the average temperature in San Francisco next week?"

Figure 4: Examples of zero-shot and few-shot prompting for tool understanding. The prompts are constructed by describing the functionalities (zero-shot prompting) or giving usage examples (few-shot prompting) of a weather API.

which requires the model to adapt to the diverse expressions of intent of different individuals. One solution is to incorporate more diverse training data that covers a wide range of real-world scenarios, thereby enabling the models to learn the nuances of different instructions. Another solution is to leverage user feedback and actively adapt the model to individual users, i.e., personalized tool learning (§ 5.4).

Tool Understanding. As noted by [Hernik & Csibra \(2009\)](#), when learning to utilize a specific tool, human beings perceive it as an object with particular functions, engaging in a cognitive process to understand its purpose and operation. By observing goal-directed demonstrations and following actions performed by other people, they gradually acquire the necessary knowledge and skills to use the tools effectively. Similarly, this understanding process is crucial for successfully solving tasks with tools. Analogously, a comprehensive understanding of the tools' functionalities is indispensable for enabling the controller to use tools proficiently. The process of tool understanding encompasses grasping what the tool is used for and how to use the tool. Take the case of a calculator: the controller needs to know that a calculator is intended for arithmetic operations, its input should be numbers and mathematical operators, and its output should be a computed value.

In real-world scenarios, tools are typically accompanied by a manual (or tutorial), which provides sufficient relevant details about their functionalities and usage. Endowed with strong few-shot learning ([Brown et al., 2020](#)) and zero-shot learning ([Wei et al., 2022a](#)) capabilities, foundation models can be prompted to unravel tools' functionalities and comprehend how to use them. To this end, we can construct suitable task-specific prompts either through manual design ([Vemprala et al., 2023](#)) or retrieval ([Zhou et al., 2023](#)). These prompts should describe the API functionalities or exemplify with demonstrations of their usage.

We categorize two prompting approaches as shown in Figure 4: (1) **zero-shot prompting**, which describes API functionalities, their input/output formats, possible parameters, etc. This approach allows the model to understand the tasks that each API can tackle; (2) **few-shot prompting**, which provides concrete tool-use demonstrations to the model. By mimicking human behaviors from these demonstrations, the model can learn how to utilize these tools. We provide experimental results of both prompting methods in § 4.

Prompting has been widely adopted as a lightweight approach to teach foundation models about tools ([Yao et al., 2022b](#); [Driess et al., 2023](#); [OpenAI, 2023](#)) with minimum human effort. Prompts can be easily adjusted to accommodate changes of tools. For instance, when tools are modified or upgraded, we can flexibly rewrite the prompts to adapt the model behaviors. Despite these advantages, prompting methods still face several challenges. First, since the effectiveness of prompting depends a lot on the model, smaller or less capable models cannot understand prompts well. Second, prompting is restricted by input context length. Although foundation models have been shown to learn to use simple tools through prompts, the situation may be more

challenging with multiple complex tools with long descriptions. Especially when the tool set greatly expands, providing all possible tools within a prompt becomes infeasible given the limited context length.

A potential solution is to add an intermediate stage of tool selection, which first retrieves a small set of tools that are most suitable for the task at hand. Another solution is fine-tuning, which optimizes models with concrete tool-use examples to understand tools. This process involves leveraging the rich knowledge obtained from human tool-use experiences. When deployed in practice, a fine-tuned model alleviates the need for incorporating tool definitions in the input, which shrinks the input length and accelerates model inference. However, a major limitation of fine-tuning is that it often requires extensive human annotations or demonstrations. Additionally, tools are frequently updated, and fine-tuned models will need to be retrained with updated examples, incurring additional costs. More discussion about the tuning-based solution is left in § 3.3.

3.2.2 Planning with Reasoning

As discussed in § 3.2.1, understanding the intent and tools lays a solid foundation for planning. Nevertheless, it is still insufficient for tackling intricate tasks. The user query q often implies a complex task that should be divided into multiple sub-tasks with proper sequencing, thereby necessitating a process of *reasoning*.

Recent research has revealed that reasoning capabilities can emerge when foundation models are scaled up to a certain size (Wei et al., 2022b). In particular, foundation models with tens or hundreds of billions of parameters can generate intermediate reasoning traces during complex problem-solving, thereby significantly enhancing their zero-shot and few-shot performances (Nakano et al., 2021; Nye et al., 2021; Wei et al., 2022b). The reasoning ability observed in the foundation models appears to facilitate a transition from System 1 to System 2 (Kahneman, 2011), enabling the accomplishment of more complex tasks.

Eliciting Reasoning in Foundation Models. Despite the extensive study of the concept of *reasoning* in the psychology literature (Wason, 1968; Kelley, 2013), the notion of *reasoning* as applied to foundation models is not clearly defined. However, in general terms, the reasoning ability in the literature of foundation models can be framed as the capacity to decompose a complex problem into sub-problems and solve the sub-problems step-by-step (Wei et al., 2022c; Press et al., 2022; Khot et al., 2022). Here we keep consistent with these works and discuss reasoning in the sense of problem decomposition and sub-problem solving.

The vanilla few-shot prompt learning (Brown et al., 2020), whereby models are provided with a prompt consisting of several examples for the given task, has been shown to fail when it comes to problems that require complex reasoning (Creswell et al., 2022). To address this issue, Wei et al. (2022c) propose Chain-of-Thought (CoT) prompting. Unlike vanilla few-shot prompt learning, CoT additionally inserts the reasoning trace required to derive the final answer for each example in the prompt. In this way, CoT prompts models to generate their “thoughts” on the necessary intermediate steps before arriving at the final answer. CoT has been proven to significantly boost performance on a wide range of tasks, including arithmetic reasoning, commonsense reasoning, and symbolic reasoning (Wei et al., 2022c).

In light of the remarkable reasoning abilities of foundation models, recent research has made successful attempts to employ them in the controller in tool learning. It is demonstrated that their reasoning capabilities enable the controller to effectively decompose a complex problem into several sub-problems, and determine which tool to call upon for each sub-problem. We categorize relevant research into two streams: *introspective reasoning* and *extrospective reasoning*. The former involves generating a static plan of tool use without interacting with the environment \mathcal{E} , while the latter generates plans incrementally by iteratively interacting with \mathcal{E} and utilizing feedback obtained from previous executions. As shown in Figure 5, the environment \mathcal{E} is invisible to the controller \mathcal{C} in introspective reasoning but is visible in extrospective reasoning, creating a closed-loop interaction among the four components.

Introspective Reasoning. This kind of reasoning directly generates multi-step plans for tool use without knowing intermediate execution results. For instance, Huang et al. (2022a) investigate the planning ability of foundation models and show that they are capable of decomposing high-level tasks into semantically plausible sub-plans. Another representative work is Program-Aided Language Models (PAL) (Gao et al., 2022), which prompts models to generate Python codes for intermediate reasoning steps. PAL uses the Python program interpreter as the tool, enabling the model to act as a programmer writing detailed comments, and achieving significant improvements in arithmetic, symbolic, and algorithmic reasoning. Notably, the idea of model-as-programmer has also been shown to be successful in embodied agents, as evidenced by ProgPrompt (Singh et al., 2022) and Code-as-Policies (Liang et al., 2022a), which prompt models to generate

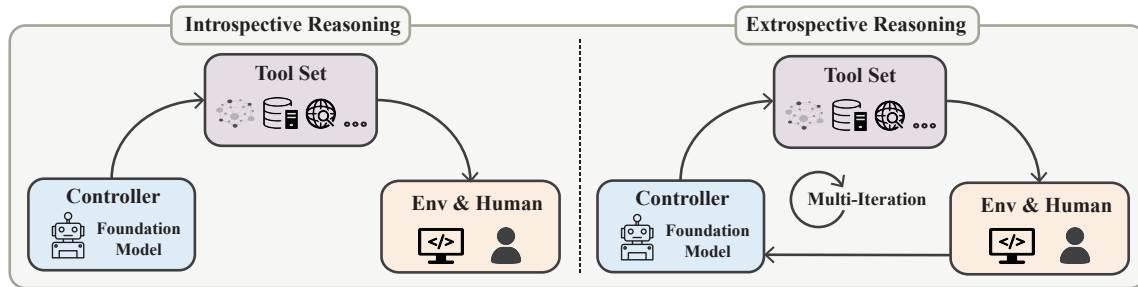


Figure 5: Illustration of introspective reasoning and extrospective reasoning. Extrospective reasoning requires feedback from the environment and humans to carry out iterative plan generation. We omit the perceiver in the illustration for simplicity.

executable programs for embodied agents. These studies reveal that, despite not having direct interaction with the environment, models are capable of generating executable programs for agents and anticipating possible anomalies in the plan execution. Another example is Visual ChatGPT (Wu et al., 2023), which interleaves various vision foundation models with ChatGPT to enable understanding and generating images. In their system, ChatGPT serves as the core controller and makes sequential decisions. At each step, ChatGPT might call a vision model to modify an existing image or respond to the user with plain text.

However, since these models are not grounded in the environment, they may generate unrealistic and nonsensical plans. To this end, SayCan (Ahn et al., 2022) emphasizes those actions that the agent is “permitted” to execute instead of those it is “willing” to perform. In practice, they employ a value function to estimate the probability of each action being successfully executed. With this function, the agent becomes more physically grounded in the environment. Overall, despite the absence of environment feedback, foundation models exhibit a remarkable ability to plan effectively in introspective reasoning. They can anticipate potential anomalies in plan execution and adjust their plans accordingly. This ability not only enables the controller to generate executable programs but also enhances its capacity to plan for a wide range of tasks.

Extrospective Reasoning. Despite its simplicity, introspective reasoning cannot adapt the plan in response to intermediate execution results. A more rational approach to planning is taking the environment \mathcal{E} into account, and generating plans incrementally (e.g., one step at a time) with subsequent plans dependent on previous execution results. This allows the four components described in § 3.1 to be well integrated and to cooperate effectively to achieve complex tasks. We refer to such an incremental reasoning strategy as extrospective reasoning. Compared to introspective reasoning, extrospective reasoning additionally considers feedback from the user and environment (Figure 5), and is thus better suited to complex tasks (Madaan et al., 2023; Wang et al., 2023b), such as multi-step QA and embodied learning, where decision-making at each step is dependent on the preceding context.

Recent works such as Self-Ask (Press et al., 2022), ReAct (Yao et al., 2022b), and ToolFormer (Schick et al., 2023) have demonstrated that by providing access to search engine APIs, models are able to achieve improved accuracy on multi-step QA. Through CoT prompting (Self-Ask and ReAct) or fine-tuning (ToolFormer), models can learn to decompose complex questions and utilize the search API to find the answer to the first sub-question. Based on the response and the question, they can then iteratively determine the subsequent question to ask or give the final answer.

For embodied learning, while introspective reasoning methods have demonstrated the ability to generate executable programs and address potential execution anomalies, direct interaction with the environment can further enhance models’ planning capabilities. For instance, Inner Monologue (Huang et al., 2022b) leverages multiple sources of feedback from the environment, such as whether a task is completed successfully and the current scene information. In this way, models can generate more feasible plans and improve their ability of high-level instruction completion. LLM-Planner (Song et al., 2022) explicitly considers anomalies that may arise during plan execution and utilizes environmental feedback to regenerate the plan in case of execution failure, enabling models to handle exceptions appropriately. Additionally, ReAct (Yao et al., 2022b) grants models the autonomy to determine when to cease generating action tokens during planning, enabling them to reason about the current situation and develop more refined subsequent plans.

In summary, extrospective reasoning requires interaction between the controller and the environment, which is a more complex setting. However, the real-time feedback from the user and environment empowers models to have a more comprehensive understanding of the current situation, making it possible to eventually achieve long-term goals that necessitate extensive planning.

Challenges in Multi-Step Multi-Tool Scenario. Humans do not stick to only one single tool to complete complex tasks. Instead, we carefully decompose the task into several sub-tasks, select the most suitable tool for each sub-task, and gradually accomplish them step by step. As discussed above, current research has shown satisfactory performance in task decomposition. However, there is a lack of exploration in utilizing different tools for different sub-tasks. Most of the research mentioned in this section is limited to either multi-step single-tool or single-step multi-tool scenarios. However, there has been a recent emergence of research that addresses the multi-step multi-tool scenario. One such example is the ReAct model (Yao et al., 2022b), which integrates multiple APIs of Wikipedia and employs the foundation model to decide when to use which API. Later, MM-ReAct (Yang et al., 2023) generalizes ReAct to the multi-modal domain by including several vision experts. Furthermore, some recent projects such as Auto-GPT⁴ demonstrate the huge potential of GPT-4 in manipulating multiple tools and making long-term plans, pushing the boundaries of what is possible with tool learning. Given a user query, Auto-GPT will take step-by-step actions to accomplish the objective autonomously. In addition to reasoning about the current state, Auto-GPT can also reflect on past actions to refine decision-making. Although these works constitute a significant step in advancing tool learning in the multi-step multi-tool scenario, there are still several challenges and future directions that need to be investigated.

- **Understanding the Interplay among Different Tools.** The multi-step multi-tool scenario typically involves complex tasks that require a higher level of intent understanding and reasoning capability. To effectively utilize multiple tools under this scenario, models need to grasp not only the individual functionalities of tools but also their interactions and dependencies. Models should be able to sequence the tools in a logical order so that the subsequent tools can leverage the information generated by the previous tools and effectively complete the task.
- **From Sequential Execution to Parallel Execution.** Tool executions do not have to be performed sequentially. In certain scenarios, parallel execution is possible for sub-tasks that do not depend on each other, which can potentially improve execution efficiency. For instance, given a user instruction “Generate two codes, one for drawing a rectangle, and one for drawing a circle.”, the two tasks can be assigned to two agents, enabling the codes to be generated simultaneously. Determining the dependencies among different sub-tasks and effectively switching between parallel and sequential execution to optimize the overall process is a promising direction that merits further investigation.
- **From Single-agent Problem-Solving to Multi-agent Collaboration.** Previous works typically assume that a single agent (controller) is solely responsible for the entire tool learning procedure. However, in practice, complex tasks often demand collaboration among multiple agents, each possessing unique abilities and expertise. Embracing multi-agent collaboration can unlock more effective and efficient problem-solving approaches, necessitating the design of methods for communication, coordination, and negotiation among agents to ensure seamless collaboration and optimal task execution. Notably, recent work like Park et al. (2023) demonstrates that multiple agents modeled with foundation models can simulate human behaviors (e.g., interpersonal communication) in interactive scenarios. This provides promising evidence for the adoption of multiple agents for tool learning.

We look forward to more work in the future moving towards more practical multi-step multi-tool scenarios and making efforts to address these challenges. As a prior exploration, we evaluate foundation models’ performance when multiple tools (APIs) are required to solve a task in § 4.

3.3 Training Models for Improved Tool Learning

Guidance, either from humans or environments, plays a critical role in training foundation models to use tools. In contrast to the prompting-based methods mentioned in § 3.2.1 and § 3.2.2, which rely on the frozen foundation models’ in-context learning abilities, the training-based method optimizes the model with supervision. As noted by Fagard et al. (2016), there are two primary ways for infants to learn a new tool, that is either from demonstration by an adult modeling the action or relying on their own exploration. Analogously, as shown in Figure 6, we categorize training strategies for tool learning into two streams: (1) learning from

⁴<https://github.com/Torantulino/Auto-GPT>

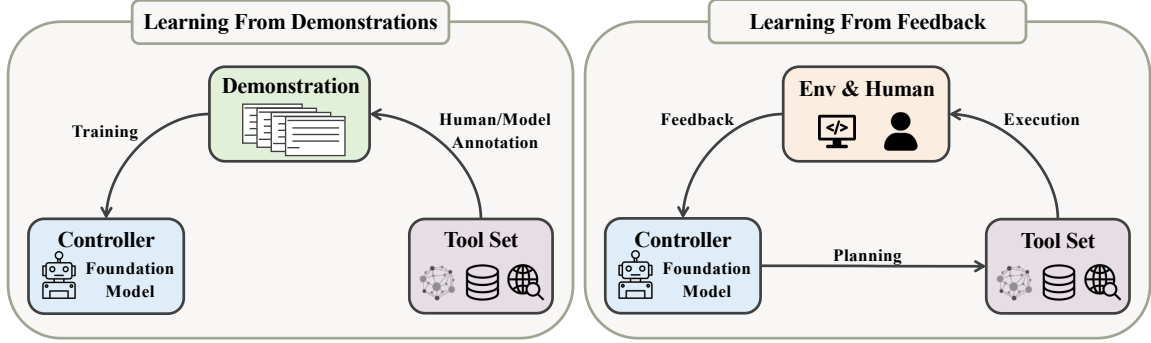


Figure 6: Training strategies for tool learning: (left) learning from human-annotated or model-annotated demonstrations; (right) learning from feedback, where the supervision could come from either the environment or humans.

concrete tool-use demonstrations (Nakano et al., 2021; Sasaki & Yamashina, 2021), which often requires human annotation, and (2) learning from feedback, which typically involves reinforcement learning (Reddy et al., 2020; Baker et al., 2022). Finally, considering the existence of potentially massive tools, learning each of them one by one is infeasible in practice. Hence, we emphasize the importance of generalization in tool learning and discuss potential solutions (§ 3.3.3).

3.3.1 Learning from Demonstrations

Models can be trained to mimic the behavior of human experts through imitation learning (Hussein et al., 2017; Liu et al., 2018b; Baker et al., 2022). Behavior cloning (Bain & Sammut, 1995) can be viewed as a simplistic form of imitation learning that focuses on learning policies in a supervised fashion, with the general assumption that the expert’s behavior is optimal or near-optimal. The objective of behavioral cloning is to train models to imitate human experts’ actions given certain inputs or conditions, and this approach is commonly adopted when the actions of an expert can be easily recorded and utilized for learning (Torabi et al., 2018).

Formally, assume that we have a dataset \mathcal{D} of size N consisting of pairs of user query q and the human demonstration annotation a^* , i.e., $\mathcal{D} = \{(q_i, a_i^*)\}_{i=0}^{N-1}$. Learning from human demonstrations optimizes the controller’s parameters θ_c with the following objective:

$$\theta_c^* = \arg \max_{\theta_c} \mathbb{E}_{(q_i, a_i^*) \in \mathcal{D}} \prod_{t=0}^{T_i} p_{\theta_c}(a_{i,t}^* | x_{i,t}, \mathcal{H}_{i,t}, q_i), \quad (3)$$

where $a_{i,t}^*$ is the human annotation at the t -th iteration for handling q_i , and T_i is the total iteration number of a_i , other variables follow the notations defined in Equation (1). Based on how a^* is obtained, learning from demonstration can be categorized into three streams, with human intervention gradually becoming less:

Supervised Learning. Traditionally, behavior cloning has been widely explored in learning end-to-end or modular perceiver-controller models for autonomous vehicles and robotic applications (Ly & Akhloufi, 2020; Codevilla et al., 2019). Recently, there has been a surge of interest in fine-tuning foundation models to perform tool-oriented tasks in a supervised way. For instance, Li et al. (2022) utilize foundation models as policy networks, whose input is the tokenized environment observations, the original goals, and action history. Benefiting from the task-general inductive bias brought by foundation models, behavior cloning using the policy network significantly improves both in-domain performance and out-of-distribution generalization. Another example is WebGPT (Nakano et al., 2021), which interacts with a search engine by iteratively refining its search queries and recording important information. To achieve this, the authors first build a search interface backed up by Bing⁵ and then fine-tune GPT-3 (Brown et al., 2020) to clone human web search behaviors. As a language model pre-trained on general domains, the original GPT-3 is not intrinsically anchored to valid browser commands. Therefore, it is crucial to first gather demonstrations of human interactions with the browser and then learn state-to-action mappings. After fine-tuning, the model shows exceptional capabilities in manipulating search engines for information retrieval, even surpassing human experts. Similarly, WebShop (Yao et al., 2022a) provides a web-based interactive environment where an agent could browse and

⁵<https://www.microsoft.com/en-us/bing/apis/bing-web-search-api>

purchase products. Through behavior cloning, the trained agent exhibits non-trivial performance in purchasing the right product given human instructions.

Semi-supervised Learning. As is often the case, human behaviors cannot be easily recorded due to time and cost considerations. However, large-scale unlabeled data is often attainable, from which we could potentially construct weak, noisy supervision. Notably, recent works have shown that we could employ a less-capable model to annotate pseudo-labels on unlabeled data and convert them into weakly-supervised tool-use demonstrations. For instance, with a small amount of seed labeled data, [Baker et al. \(2022\)](#) train a model to predict pseudo-labels of the action taken at each timestep in a Minecraft video game. Learning from these pseudo-labels, a more powerful model can be trained without requiring the rollout of models in a target environment or large-scale gold-standard human behavior annotation.

Self-supervised Learning. Despite reducing the heavy requirements on human behavior annotation, semi-supervised learning still requires a seed labeled dataset to attain the pseudo labels. Besides, the biases in the seed dataset may also be amplified during training, leading to poor generalization performance. To this end, researchers recently show that with a few demonstrations, foundation models can teach themselves how to utilize a tool in a self-supervised manner ([Parisi et al., 2022](#); [Schick et al., 2023](#)). For instance, Toolformer ([Schick et al., 2023](#)) leverages the in-context learning ability of foundation models to iteratively bootstrap tool-use examples based on a handful of human-written examples. These auto-generated examples are further filtered to reduce noise. The final tool-use dataset contains sufficient supervision, significantly improving GPT-J’s ([Wang & Komatsuzaki, 2021](#)) tool-use performance, highlighting the potential of self-supervised learning for enhancing tool-use capabilities.

3.3.2 Learning from Feedback

Collecting manually annotated tool-use examples, which probably include complete traces of human behaviors and the final answers, is time-consuming and labor-intensive. Moreover, the learned model may not adapt effectively to new environments as it conforms to the recorded human behaviors. Besides, it is impractical to explicitly annotate every possible scenario of environment condition and agent behavior ([Codevilla et al., 2019](#)). Alternatively, humans learn from trial and error to correct and rectify their tool-use behaviors ([Allen et al., 2019](#)). Similarly, feedback from both the environment and humans can enable the model to understand the consequences of its actions and adapt its behaviors. The supervision from feedback can also enhance the capabilities of an agent trained in a supervised way ([Nakano et al., 2021](#); [Baker et al., 2022](#)). Formally, learning from feedback can be described as optimizing the controllers’ parameters θ_C from open explorations with query set $Q = \{q_i\}_i$:

$$\theta_C^* = \arg \max_{\theta_C} \mathbb{E}_{q_i \in Q} \mathbb{E}_{\{a_{i,t}\}_{t=0}^{T_i} \in p_{\theta_C}} \left[R(\{a_{i,t}\}_{t=0}^{T_i}) \right], \quad (4)$$

where R is the reward estimated from the sequence of feedback and T_i denotes the number of iterations needed for handling q_i .

Reinforcement Learning (RL) for Tool Learning. RL is a common solution to enabling artificial agents to learn from their environment in complex decision-making processes ([Silver et al., 2018](#); [Berner et al., 2019](#); [Schrittwieser et al., 2020](#)). Tool learning can be considered an RL scenario, where the action space is defined by tools, and the agent learns to select the appropriate tool and perform the correct actions that maximize the reward signal. The policy model can be initialized by a foundation model ([Schulman et al., 2017](#)). Such initialization brings the policy model abundant prior knowledge, alleviating the need for the RL agent to learn basic skills. With a reward function that quantifies the performance of the agent in achieving the task goal, RL has been successfully used in various tool learning scenarios, such as robotic grasping ([Levine et al., 2018](#)) and multi-agent autotutorials ([Baker et al., 2020](#)). By optimizing the loss function, the agent learns to reflect on the current state of the environment, select the appropriate tool, and perform the right actions that lead to the highest expected reward. In the following, we introduce two sources of feedback: environment feedback and human feedback, which can be considered sources of reward signals in the context of tool learning. These two feedbacks are complementary and can be combined with each other.

Environment Feedback. The controller interacts with the environment and receives feedback about the consequences of its actions. The model then updates its policy based on this feedback to improve its tool-use behavior. Environment feedback can be categorized into two forms: (1) **result feedback**, which is ultimate feedback returned from the environment, indicating whether the model’s actions have successfully completed

the task or not. This type of feedback performs an overall assessment of the planning generated by the model. For instance, WebShop (Yao et al., 2022a) uses a hand-coded reward to assess the similarity between human-bought and model-bought products, which indicates whether the actions performed by the controller lead to the correct final product. By receiving feedback on the success or failure of its actions, the model can iteratively update its planning strategy, and adjust its decision-making process; (2) **intermediate feedback**, which refers to the state change of the environment triggered by an action. By observing the state changes, foundation models can learn whether each action is effective and appropriate, making the model better adjust its behaviors accordingly. This kind of feedback provides more detailed and timely information about the effectiveness of each tool execution. Take the case of interacting with a search engine to gather information for question-answering, models could update their policy for more efficient information retrieval by observing the rendered information of a search query.

Human Feedback. Humans could give the model rewards and penalties based on its generated plans to regulate its behavior. Human feedback can be **explicit**, which provides clear and direct insights into the model performance representing human preferences. For example, rating the quality of the model-generated action on a scale of 1 to 5; human feedback can also be **implicit**, which is not directly specified by the user but can be derived from user behavior and interactions with the model. Examples include users’ comparison (Ouyang et al., 2022), response time, and actions taken after receiving a model’s output (e.g., clicking on a recommended link).

Though human feedback is accurate and stable, it is label-intensive and has high latency. To address this issue, reinforcement learning from human feedback (RLHF) (Christiano et al., 2017) is proposed to finetune a model to imitate humans to give rewards, which are then used to optimize the policy with RL algorithms such as PPO (Schulman et al., 2017). RLHF has yielded exceptional performance in various domains such as text summarization (Ziegler et al., 2019; Stiennon et al., 2020). RLHF can also improve a model’s tool-use capabilities even if it has been trained on sufficient supervised human demonstrations. For instance, WebGPT (Nakano et al., 2021) utilizes human feedback to guide a policy model to align with human preferences, which helps better manipulate search engines to answer long-form questions.

Despite its remarkable performance, RLHF still faces challenges: (1) **task-specific nature**: the corresponding evaluation criteria for specific tasks need to be pre-defined, and the preference data annotated for one task is hard to be transferred to other settings, which limits the applicability of RLHF to a wider range of tasks. To this end, it is critical to develop a universal reward model that generalizes to various tasks; (2) **biases**: RL agents optimize towards the pseudo-human reward model, thus can be up-bounded and biased by human preferences. Besides, societal biases or personal experiences may be amplified during RLHF, and it is essential to carefully evaluate the learned reward model for any biases and take measures to mitigate them.

3.3.3 Generalizable Tool Learning

Generalization of tool use is a key characteristic of human intelligence (Seed & Byrne, 2010; Teschke et al., 2013; Osieurak et al., 2018). The ancient human, for instance, recognized that regardless of the specific tool being used, a sharp edge was essential for achieving clean cuts and efficiently carrying out tasks. This recognition allowed them to transfer their knowledge of sharpening a knife to sharpening other tools, such as scrapers or choppers. Generalization is also a critical aspect of tool learning, especially considering the existence of a massive and rapidly expanding array of tools. Although conducting supervised fine-tuning on a vast collection of tool-use data can be a potential solution to facilitating generalization, collecting enough supervised tool-use data and ensuring its quality and diversity is time-consuming and practically infeasible.

Generalizable tool learning highlights the importance of *abstraction*, which is the process of identifying the essential features of a tool. Abstraction involves recognizing commonalities and patterns of tools so that models could synthesize and transfer their knowledge and skills, enabling them to use novel tools with ease. For instance, by abstracting essential features such as layers, filters, and color adjustments, users can transfer their knowledge of using Adobe Photoshop to Adobe Illustrator, even if the interface and specific tool names in these two figure-editing software are different. Abstracting these general features of tools can quickly help users learn a new tool effectively by building on previous experience.

Foundation of Generalization: Interface Unification. To facilitate knowledge transfer among tools, it is critical to design a unified interface that enables the model to manipulate various tools in a consistent and standardized manner, which serves as the foundation for generalizable tool learning. Through a unified interface, models can identify and abstract essential features of tools more easily in a unified tool protocol rather than grappling with the difficulty of understanding various tool interfaces. Currently, the manipulation

of tools is through predicting discrete action tokens, and the action space is not aligned in different scenarios, which prohibits the models from quickly adapting to new scenarios and tools. Inspired by the aspect we categorize tools in § 2.2, we identify three potential ways of interface unification: the semantic interface, GUI interface, and programming interface.

- **Semantic Interface.** The semantic interface operates by utilizing a specific text span (action name) as the action trigger, which is the most intuitive and natural way for interface unification. For instance, ReAct (Yao et al., 2022b) employs `Action:Search` as the trigger for the function that searches for relevant passages. In robotic manipulation (Ahn et al., 2022; Liu et al., 2023), the generated natural language (e.g., `pick up the sponge`) is mapped to specific actions. Despite its ease of implementation, the semantic interface poses certain challenges that must be addressed. First, the mapping between the generated text and the corresponding tool action should be pre-defined individually, which is a laborious task, particularly when the tool set expands quickly. Moreover, the model may fail to accurately produce the precise form to trigger the intended action, even leading to false triggering of actions.
- **GUI Interface.** Humans primarily interact with the digital world through GUI interface (e.g., mouse and keyboard), which has been extensively optimized to follow human action efficiently. Nevertheless, before robots can learn to use a GUI interface flexibly, it is necessary to establish a virtual environment that can facilitate mapping predicted tokens to human-like mouse movements and keyboard inputs. Prior research has explored providing platforms for agents to complete web-based tasks using keyboard and mouse actions (Shi et al., 2017; Liu et al., 2018a). However, these environments restrict models to a limited set of pre-defined mouse options and common keyboard actions such as copy and paste. By leveraging foundation models, it is possible to introduce prior knowledge regarding common combinations of keyword and mouse actions, thereby expanding the potential actions that a model can execute.
- **Programming Interface.** This kind of interface allows the model to go beyond pure natural language and specify its action using a program. Such unification requires the model to be acquainted with the syntax of the function calls. The recent code-generating language models (CLM) such as InCoder (Fried et al., 2022) and CodeX (Chen et al., 2021) provide the possibility of such unification. The programming interface has been applied widely. For example, Code-as-Policies (Liang et al., 2022a) finds that with CLM as the backbone for robotic control, the robots can leverage the code grammar to execute complex actions, generalize to novel instructions, and give precise control with accurate parameter values to the functions. The programming interface provides promising opportunities for tool learning because (1) complex tool learning logic can be modeled using the control flow of programming language; (2) explicit calls of external APIs can be naturally implemented by executing programs.

It should be noted that the interface selection should align with the capabilities and limitations of the foundation model. For instance, language foundation models are trained to generate text and may be better suited for the semantic interface. Similarly, a multimodal foundation model that combines visual and textual information may be more appropriate for the GUI interface, as it can understand and generate human-like mouse movements and keyboard inputs. On the other hand, code foundation models may be more suitable for the programming interface, as it is trained to understand code syntax and function calls.

Under certain cases, we may face challenges where the tool’s output is not aligned with model’s input format. A common practice is to compose the functionality of the model and tool in the same modality. For example, Zeng et al. (2022) chain together foundation models of various modalities by converting their outputs into natural languages. This simple method leverages prompting to compose new multimodal capabilities without fine-tuning. In contrast, another solution is to building multimodal foundation models that can perceive general modalities, based on the belief that multimodal foundation models can all be unified through a general-purpose interface (Alayrac et al., 2022; Hao et al., 2022). Gato (Reed et al., 2022) is a representative generalist multi-embodiment agent trained on tremendous datasets of agent experience. Gato can sense and act with different embodiments, such as playing Atari, captioning images, chatting, etc. Similarly, PaLM-E (Driess et al., 2023) incorporates continuous inputs from different modalities into a PLM. By joint training on multiple embodied tasks, PaLM-E could make grounded decisions in the real world.

Strategies of Generalizable Tool Learning. In general, a unified interface enables models to learn and transfer knowledge more easily and efficiently, but it does not guarantee optimal learning outcomes in all scenarios. Generalizable tool learning requires models to further adapt, refine, and specialize their learned knowledge to specific tasks or domains. Here, we discuss two potential approaches to achieving this goal and facilitating generalization.

- **Meta Tool Learning.** Metacognition (Clarebout et al., 2013) is a crucial aspect of human intelligence that allows individuals to reflect on their own thinking and adapt their behaviors when faced with unfamiliar situations. In the context of tool learning, metacognition refers to the ability of a model to reflect on its own learning process and adapt new tool-use strategies when necessary. With metacognition, models can identify common underlying principles or patterns in tool-use strategies and transfer them to new tasks or domains. Take the case of the web search tool, when the model trained on a source search engine (e.g., Bing Search) is transferred to a target one (e.g., Google Search), the model can leverage its metacognitive awareness to adapt its tool-use strategies based on its previous experiences. This may include identifying common underlying patterns in tool-use strategies, such as effective search queries, relevant results, and user feedback, and using this metacognitive awareness to better align with the algorithms and user interface of the new search engine.
- **Curriculum Tool Learning.** Another approach to improving model generalization is through curriculum learning (Bengio et al., 2009), which starts with simple tools and gradually introduces the model to more complex tools so that it can build upon its prior knowledge and develop a deeper understanding of the tool. For instance, we could start with a curriculum of basic algorithms and operations to effectively teach a model to use Mathematica⁶, e.g., addition and subtraction, and then gradually move on to more complex mathematical concepts like calculus and linear algebra. This training strategy ensures that the model is introduced to the simple, essential features of the tool before moving on to more complex concepts in a way that is manageable and effective. Moreover, curriculum tool learning allows the model to learn how complex tools are built upon simple tools. It provides an understanding of how a complex tool can be seen as an updated high-level version of a simple tool, and how its function is a combination of several basic tools. This understanding of the relationship between simple and complex tools facilitates the transfer of previously learned knowledge to new tools, enabling the model to more effectively identify similarities and differences between situations and adjust its approach accordingly.

4 Application and Experiment

In this section, we aim to explore the applications of tool learning and investigate the efficacy and limitations of state-of-the-art foundation models in utilizing tools. We select 18 representative tools for evaluation and place the main results in this section. For more case studies of ChatGPT, please refer to Appendix A.

4.1 Evaluated Tools

We first briefly introduce the tools selected in experiments as follows:

Machine Translator. General-purpose language models may exhibit suboptimal proficiency when processing text from multiple linguistic domains. Machine translators can effectively alleviate this issue by enabling non-translation-dedicated language models to better comprehend multi-lingual texts. Following Toolformer, we use NLLB (Costa-jussà et al., 2022) as our translator and choose MLQA (Lewis et al., 2020a), a multilingual question answering benchmark, as the testbed. Given a context in English and a question in Arabian, the task requires answering the question using English. We randomly sample 200 test instances from the original test data. For the evaluation metric, we choose F1-score.

Calculator. Following the setting of Toolformer, we conduct experiments in which language models use a simple calculator to solve math word problems. We choose a simple implementation for the calculator, which supports basic arithmetic operations (i.e., +, −, ×, ÷). We evaluate two math word problem datasets: ASDiv (Miao et al., 2020) and MathQA (Amini et al., 2019) and choose accuracy as the metric.

Map. We choose Bing Map API⁷ for location information retrieval, assisting in user queries related to the route, driving distance, latitude coordinates, nearby locations of interest, etc. We manually curate user queries through crowdsourcing.

Weather. We choose Weather API⁸ and investigate whether models could use the tool to answer weather-related questions, such as questions about current weather in any city, forecasting the weather within two weeks in any city, and giving suggestions based on the weather information. Two APIs are supported, the first one is `GetWeatherToday<city>`, which provides the current weather condition of a city; another

⁶<https://www.wolfram.com/mathematica>

⁷<https://learn.microsoft.com/en-us/bingmaps>

⁸<https://www.weatherapi.com>

one is `ForecastWeather<city, N>`, which forecasts the weather of a city after N days. The detailed information returned includes the temperature, wind speed, UV index, sunrise, sunset time, etc. We manually curated 100 weather-related user queries.

Stock. We choose Alpha Vantage Stock API⁹ for stock market querying. We aim to obtain specific information about the opening, closing, highest, or lowest price for one particular stock on one specific day or month. We manually curate 1200 question-answer pairs and choose accuracy as the evaluation metric.

Slides. Slides-making is traditionally performed by humans using a human-computer interface (e.g., mouse and keyboard). However, current models cannot directly move a mouse or press computer keys. To address this limitation, we provide six APIs with high-level semantics for the model. Four APIs are built based on the open-source library `python-pptx`¹⁰ to control the slides, one API allows the model to retrieve images from the internet based on a topic, and one API is used to submit and display the final slides to the user. To collect the data, we brainstorm 50 different careers that require slides-making in their work, for each career, we brainstorm 2 cases where practitioners have to make slides. The final dataset consists of 100 slides-making tasks. We evaluate the model’s performance by counting the fraction of instances in which the model-generated API calls are correctly executed without errors.

Table Processing. We craft a suite of table processing APIs using `pandas.DataFrame` in Python. By leveraging these tools, models can provide a more natural and streamlined experience for users, allowing them to perform data analysis and visualization tasks directly. We manually construct a table processing dataset containing 13 tables and 117 corresponding queries.

Knowledge Graphs. Knowledge graphs contain factual knowledge about the real world, which is stored in the form of RDF triplets. The triplets can be retrieved by SPARQL (Standard Protocol and RDF Query Language). We provide 7 APIs that mimic the process of human querying the knowledge graph, including showing the candidate entity/relation given a name surface form, showing a head entity’s home page, showing a tail entity’s home page, sending SPARQL queries, showing the result of SPARQL queries, and finding a keyword in the output of a SPARQL query. We curate 64 questions that could be answered by querying knowledge graphs.

Search Engine. We choose Bing Search API¹¹ and test the model on real-time question answering. Two APIs are supported: the first one is `Search<query>`, which returns the top-related search results back to the model; another one is `LoadPage<N>`, which loads the detailed information of page N indexed in the search results, and returns the detailed contents. We experiment with RealTimeQA (Kasai et al., 2022), which is a dynamic question-answering platform that inquires about novel events or information. Specifically, we choose the most recent release (20230217 version) of multiple-choice data for evaluation. Given the question and choices, the model is expected to interact with the search engine to extract the necessary information, before settling on the final answer.

Wikipedia. We largely build our Wikipedia Search tool upon ReAct (Yao et al., 2022b) with slight modifications on the API designs. The tool consists of 3 APIs: `search<entity>`, which searches for an exact entity name on Wikipedia and returns the first 5 sentences of the corresponding page if the entity exists; otherwise, it displays related entity names; `lookup<keyword>`, which looks up the keyword on the current page and returns the next sentence containing the keyword, similar to humans’ using the CTRL+F function on a web page; `disambiguate<entity>`, which inputs an entity name and displays all entities that share the same name. We focus on HotpotQA (Yang et al., 2018b) for question answering. We conduct our experiments in an open-domain setting, where only the question is shown to the model. We randomly sample 200 instances from the dataset.

Online Shopping. Amazon online shopping is a relatively complex web environment, in which models need to buy a commodity that satisfies various requirements mentioned in a user instruction. Based on WebShop (Yao et al., 2022a), we build our online shopping tool, which covers mainstream online shopping actions including searching for an item, loading detailed information about an item, choosing a feature for an item, going to the previous/next page, deciding to purchase, etc. We use the dataset provided by WebShop and randomly sample 100 test instances, which cover instructions about various customers’ needs with specific requirements of commodities’ attributes.

Embodied Scene. ALFWorld (Shridhar et al., 2021) is an aligned text and embodied environment game, where agents need to interact with objects (e.g., fridge, microwave, drawer, etc.) in a house to complete a task

⁹<https://www.alphavantage.co/documentation>

¹⁰<https://pypi.org/project/python-pptx>

¹¹<https://www.microsoft.com/en-us/bing/apis/bing-web-search-api>

Tools	# APIs	Test Set	Test Size	No Tool	Zero-shot	Few-shot
Machine Translator	1	MLQA	200	49.1	49.7	54.0
				38.2	38.6	45.5
Calculator	1	ASDiv	266	85.3	81.6	92.5
				91.7	74.1	92.5
Map	11	Curated	129	—	58.1	86.8
				—	29.7	86.8
Weather	2	Curated	100	—	39.0	99.0
				—	92.0	99.0
Stock	8	Curated	122	—	33.6	63.1
				—	39.0	64.8
Slides	6	Curated	100	—	95.0	97.0
				—	94.0	86.0
Tables	21	Curated	117	54.8	60.7	85.2
				60.9	73.0	92.2
KGs	7	Curated	64	—	42.2	46.9
				—	7.8	14.1
Search Engine	2	RealTimeQA	30	50.0	50.0	66.7
				50.0	43.3	63.3
Wikipedia	3	HotpotQA	200	33.5	28.5	35.5
				34.5	8.5	19.0
Online Shopping	2	Webshop	100	—	38.4	37.1
				—	42.0	35.9
Embodied Scene	1	ALFWorld	134	—	51.0	78.0
				—	23.0	81.0
Cooking Assistant	3	Curated	50	—	84.0	98.0
				—	82.0	90.0
Movie Search	3	Curated	60	—	77.0	72.0
				—	43.0	75.0
AI Painting	2	Curated	25	—	93.0	100.0
				—	90.0	88.0
3D Model Construction	14	Curated	10	—	20.0	40.0
				—	0.0	40.0
Chemical Properties	4	Curated	100	35.0	55.5	73.5
				46.5	67.0	81.0
Database	4	Curated	50	—	50.0	75.0
				—	58.3	75.0

Table 1: We list the overall results of different tools evaluated in this paper. “# APIs” denotes the number of APIs corresponding to each tool. The test set means the dataset we employed in conducting the experiments. We show the result of three settings i.e., **No Tool**, **Zero-shot**, **Few-shot**. The results of text-davinci-003 are shown on white background, while the results of ChatGPT are shown in cyan background.

(e.g., putting a clean spatula in a drawer). We largely follow the setting of ReACT (Yao et al., 2022b) and report the success rate on the valid set.

Cooking Assistant. We choose AllRecipe¹² to investigate whether models can find the proper cooking recipe and extract important details. The tool is designed similarly to the search engine tool. With this tool, the model

¹²<https://www.allrecipes.com/>

can perform: (1) finding the target recipe, and (2) answering questions based on observed details. We manually curate 50 queries for evaluation.

Movie Search. We choose Douban Film API¹³ to search for movie-related information. Three APIs are devised with the aim of discovering movies that are currently playing or upcoming, as well as extracting detailed information about each movie. We curate 60 questions about the movies, such as recommending some movies which are on display or upcoming and providing a brief introduction to a movie.

AI Painting. AI image generation model has been widely used by human artists. To endow models with the capacity to create images using the AI image generation model, we provide the following APIs: one API generates an image given a prompt using stable diffusion (Rombach et al., 2022), others are the image segmentation¹⁴ and image inpainting APIs¹⁵, which replace a target object in an image with a new object described by a prompt. We curate 25 queries as the initial prompt, together with subsequent queries for modifying that image.

3D Model Construction. We investigate three-dimensional (3D) modeling by manually devising a collection of APIs that leverage the capabilities of the sophisticated 3D rendering engine Taichi¹⁶. Due to the complexity of executing this API (3D rendering), we only demonstrate the performance on 10 curated questions.

Chemical Properties. To evaluate the capability of tool learning in professional domains, we utilize the Chemical Property query, and more specifically, the PubChem¹⁷ API for resolving scientific inquiries. 4 APIs are supported, which facilitate the retrieval of a chemical’s identification number based on the name or SMILES notation (Weininger, 1988), as well as obtaining the chemical’s properties based on its identification number. We manually curated 100 questions for evaluation.

Database. We explore the potential of tool learning in accessing database data via natural language. The fundamental APIs include (i) obtaining the structural information of the target data (schema); (ii) translating the input natural language text into an equivalent SQL query; (iii) rewriting the query into an execution-efficient one; and (iv) querying the result data by connecting to the database. We automatically curated 50 relatively complex queries under the TPC-H schema, which are of relatively complex structures (involving 2-6 tables and composite predicates) and take over 111.5 minutes to execute in total. The evaluation metric is the ratio of queries for which tool learning can accurately output the results.

To facilitate future research attempts, we implement and integrate all the above tools into **BMTools**¹⁸, which is an open-source repository that extends foundation models using tools and also serves as a platform for the community to build and share tools. With BMTools, users can easily build a new plugin by writing Python functions and also integrating external tools from other sources (e.g., ChatGPT Plugins).

Building a *tool library* for foundation models is critical to connecting foundation models with tools and we are glad to see there are emerging works in this direction. LangChain¹⁹ is the first open-sourced project that attempts to chain foundation models with tools. Under a unified interface, users could either build their own task pipelines or let the foundation models call APIs. Most recently, TaskMatrix.AI (Liang et al., 2023) and HuggingGPT (Shen et al., 2023) extend APIs and tasks to broader scenarios, including multimodal models for visual tasks, local software, and cloud service APIs. OpenAI also proposed its official tool library, ChatGPT Plugins²⁰, to empower ChatGPT with other applications. By simply providing APIs with descriptions, ChatGPT is enabled to call applications and complete more complex tasks. Different from third-party libraries, ChatGPT plugins are cautious about safety risks and establish strict standards for plugins. The library prioritizes the most essential tools such as the web browser, code interpreter, and retrieval plugin.

4.2 Experiments

Settings. We conduct experiments on all the above tools and choose both text-davinci-003 and ChatGPT to evaluate their performance with zero-shot prompting and few-shot prompting as mentioned in § 3.2.1:

¹³<https://movie.douban.com>

¹⁴<https://huggingface.co/CIDAS/clipseg-rd64-refined>

¹⁵<https://huggingface.co/runwayml/stable-diffusion-inpainting>

¹⁶<https://github.com/taichi-dev/voxel-challenge>

¹⁷<https://pubchem.ncbi.nlm.nih.gov>

¹⁸<https://github.com/OpenBMB/BMTools>

¹⁹<https://docs.langchain.com>

²⁰<https://openai.com/blog/chatgpt-plugins>

-
- **Zero-shot** prompting provides the instruction to model about the task description, and information about the APIs in the tool. Some basic guidelines can also be added to the instruction.
 - **Few-shot** prompting additionally adds concrete tool-use examples as a hint of how to use the APIs given a user query. Providing examples is expected to improve the performance.

Whenever feasible, we also compare the results with a baseline that does not involve the utilization of tools, i.e., **No Tool**. In such cases, we solely depend on the model’s internal knowledge to accomplish the given task (e.g., machine translation). Nonetheless, many tasks (e.g., slides-making) cannot be completed without the aid of tools. Consequently, we omit the “no tool” configuration in such cases.

In the experiment of machine translator, calculator, search engine, Wikipedia, online shopping, and ALFWorld, we employ existing datasets for evaluation. However, for other tools, a suitable dataset for experiments does not exist. To address this issue, we adopt a methodology similar to that of Wang et al. (2022b), wherein we curate a set of user queries. Specifically, we manually write a few user queries as seed examples and use ChatGPT’s in-context learning ability to generate more instances. Then we manually filter those instances with low quality. We find empirically that the generated examples are diverse enough. Unless otherwise specified, for these manually curated test sets, we employ the trace of API calls as the metric for evaluating the models’ performance. Specifically, if humans judge that all the API calls are accurate for the given task, and they yield a reasonable result, the task is deemed to be correctly completed. The codes and our curated dataset will be made available to the academic community²¹.

Results. We present the results in Table 1, from which we can conclude that: (1) In most cases, models can learn how to effectively use tools with simple prompting, and improve their task performances. (2) For the tasks that models can leverage their internal knowledge to solve (such as the cases of the calculator and search engine), utilizing tools with zero-shot prompting could sometimes lead to worse performance, which implies that sub-optimal utilization of tools may negatively impact performance. Nevertheless, incorporating tools with few-shot prompting still consistently yields superior performance than not incorporating tools. This underscores the concrete benefits that tools can bring to problem-solving, provided that they are employed effectively. (3) Additionally, comparing the performance of ChatGPT and text-davinci-003, we observe that although ChatGPT has been fine-tuned with RLHF, it does not yield better results than text-davinci-003. We attribute this to two reasons: firstly, the alignment tax issue mentioned in Ouyang et al. (2022), that is, the specific task skills and in-context learning ability are undermined during RLHF training; secondly, the model size of ChatGPT, though not officially stated, might be much smaller than text-davinci-003, thus making ChatGPT harder to handle complex scenarios.

Regarding the performance of different tools, it is important to acknowledge that the evaluation setups of these tools are inherently different, making direct comparison difficult. However, limiting our comparison to solely those tools that employ manually curated test sets and examining the successful rate of API calls, we have observed that under the few-shot prompting setting, certain tools such as Map, Weather, Slides, Tables, Cooking Assistant, and AI Painting exhibit a satisfying completion rate. These tools are deemed to be less challenging than other tools. In fact, we find empirically that both ChatGPT and text-davinci-003 can utilize these tools proficiently despite not directly being fine-tuned on them.

However, for several tools such as KGs, Wikipedia, online shopping, and 3D model construction, the model performance is still far from satisfactory even with few-shot prompting. The reason is perhaps that the usage of these tools cannot be easily learned with a few examples. For example, tools requiring the generation of executable code as the parameter to the API, such as the `search_by_query` API in the KGs tool (see Appendix A.9 for more details), are found to be significantly more arduous. This implies the necessity of training foundation models to use tools as mentioned in § 3.3. We provide the prompts and model responses of ChatGPT in Appendix A as case studies for all the tools.

5 Discussion

5.1 Safe and Trustworthy Tool Learning

Armed with external tools, AI systems can be unprecedentedly capable and human-like. With the ability to perceive, act, and make decisions, these models can potentially intervene and significantly influence human society. Although we are eager to witness how tool learning with foundation models will change our life, it is

²¹<https://github.com/OpenBMB/BMTools>

paramount to take a step back and contemplate the underlying risks. For responsible AI research, here we discuss the safety and trustworthiness problems of tool learning.

Adversaries. Same as all the other AI systems, we could foresee that there will be external adversaries once the tool learning models are deployed in reality, and thus how to defend against these threats is of great significance (Szegedy et al., 2014; Wallace et al., 2019; Jin et al., 2020; Hendrycks et al., 2021). Recent works suggest that large foundation models like ChatGPT are more robust on hard and adversarial examples (Taori et al., 2020; Wang et al., 2023a), which improves their utility in the complicated real world. But the attempt of crafting misleading or even harmful queries will undoubtedly persist as well (Perez & Ribeiro, 2022). Moreover, due to training on massive web data, foundation models are faced with long-lasting training-time security issues in deep learning, such as backdoor attacks (Kurita et al., 2020; Cui et al., 2022) and data poisoning attacks (Wallace et al., 2021).

In addition to foundation models, the incorporated tools could be new attack targets for adversaries. For example, the attackers could maliciously modify the manual documentation or even the tools themselves (e.g. attacking a news API to give biased reports) to mislead the model into erroneous outcomes. The key challenge lies in the interplay between foundation models and tools, since a safe and robust system requires the models to not only learn to use tools, but also possess the ability to scrutinize, rectify, and secure them. Currently, most research endeavors aimed at defending against external adversaries focus solely on ensuring the model safety. Nonetheless, in light of the everchanging paradigm shift, safety research must also attend to tools to protect the entire system.

Governance. There is long-standing worry about the misuse of AI, especially the powerful foundation models (Bommasani et al., 2021). Under the paradigm of tool learning, governance over foundation models is more urgently needed. The pertinent question at hand is *which tools should be involved?* In § 4, we list a bunch of tools that may empower foundation models to solve complicated tasks. However, given the countless tools human beings have manufactured, we must consider if it is appropriate to allow models to master all of them. Certain tools, such as calculators and translators, may be deemed safe as they do not pose any harm to individuals. However, granting models access to the internet or permitting them to make decisions in the real world could be perilous, as they could cause negative or even dangerous influences such as disseminating falsehoods (Zellers et al., 2019) and harming human lives. In this regard, research communities and companies need to deliberate carefully before permitting machines to master a certain tool.

Apart from potentially engaged harmful tools, governance over tool usage is also a pertinent issue. As highlighted by Amodei et al. (2016), the end-to-end training paradigm in deep learning does not regulate how models achieve their objectives. Fortunately, such goal-oriented approaches did not result in catastrophic consequences due to the capability limitation of task-specific models, but it warrants serious consideration moving forward. Foundation models are not only expected to finish tasks with the help of tools but also should follow the regulations and constraints of tool usage.

Trustworthiness. The goal of tool learning lies in creating advanced intelligent agents. However, determining whether these agents are trustworthy or not is a complex challenge. Even though tool learning delivers enhanced interpretability and robustness, the core foundation models are still considered “black boxes”. Recent research (Chen et al., 2022b) shows that although large models achieve better performance, they are unable to predict when they will be wrong, rendering the calibration problem unresolved yet. Accompanied with tools, under what circumstances will the model call on the tools is unpredictable as well. Therefore, before we apply these models to high-stake scenarios such as autonomous driving (Milakis et al., 2017) and clinical trials (Matheny et al., 2019), it is essential to thoroughly discuss to what extent should we allow AI to engage in human lives.

Moreover, the morality of foundation models has emerged as a contentious issue in recent times. Despite OpenAI’s commendable efforts to imbue InstructGPT (Ouyang et al., 2022) and GPT-4 (OpenAI, 2023) with human values and preferences, given the discomfiting “jailbreak” responses by ChatGPT (Borji, 2023) and New Bing (Roose, 2023), whether these big models will be mild and compliant remains doubtful. Ironically, the very discourse that once centered around the potential recklessness of autonomous robots is now mirrored in the development of large language models, thereby fueling a self-fulfilling prophecy that further exacerbates the already frayed trustworthiness of these systems. When models could learn actively from the world via tools, the challenge of controlling their actions will become more daunting than ever before.

5.2 Tool Learning for Large Complex Systems

Different from tools with limited functionality, large complex systems (e.g., relational databases (Zhou et al., 2020), manufacturing execution systems Kletti (2007), supply chain management systems Misra et al. (2010)) are composed of numerous components (e.g., over 500 knobs that control different functions in relational databases (Li et al., 2019)). This complexity results in much more flexible and complicated interaction and management approaches. Consequently, there are three main challenges in applying tool learning in large complex systems.

System Learning. Acquiring and comprehending the knowledge and functions of a large complex system is a labor-intensive task for human beings. Similarly, existing foundational models face challenges in memorizing and mastering the relevant skills associated with such systems. For instance, relational databases involve numerous built-in functions (e.g., selecting, inserting, updating, and managing user data), complicated query syntax (e.g., ORM code, SQL queries), and various maintenance problems during execution (e.g., connection anomalies, workload contention, resource problems). To address this issue, we first need to fine-tune foundation models with textual materials or even source code that describe the components, functions, execution mechanisms, and even anomaly cases of the system. This process deepens the models' comprehension of the intricacies inherent in the system. Next, we must carefully design prompts to assist foundation models to become proficient in using specific functions (e.g., determining the calling order) and effectively manipulating this system. Finally, existing foundation models may occasionally generate erroneous or unintended actions. We can augment these models with a checking layer (e.g., SMT solver), which ensures accurate generation of function calls by the foundation model and derives both legal and reasonable interactions with the tools.

Efficiency Requirements. In many scenarios, the efficiency of tool learning with foundation models is a critical metric when implementing inside real systems. For example, in real-time scenarios (e.g., fraud detection (Wang, 2010)), users expect fast and accurate responses within milliseconds. However, existing foundation models take a relatively long time to reason and plan how to call the functions, which is intolerable in practice. Thus, it is crucial to find efficient ways to speed up the response of tool learning while maintaining high performance. First, we can filter out irrelevant, redundant, or low-quality input information before sending it to the foundation model. Second, we can add hints in zero-prompting to enforce the foundation model to jump over redundant function calls and reasoning (e.g., ending the process right away after obtaining a good enough solution). Third, we can cache the relevant knowledge in high-speed storage (e.g., in-memory vector database) and the foundation models only retrieve the relevant knowledge to augment the reasoning procedure when necessary (e.g., the disk status information for system diagnosis).

Privacy Concerns. When training the foundation model for tool learning, it requires much user behavior data to simulate the thinking, planning, and decision-making processes. However, in modern systems, a significant portion of user data is of high sensitivity, which necessitates strict privacy and security regulations to ensure responsible data utilization. To address the privacy concerns, two possible solutions are federated learning and model distillation. First, federated learning enables the training of foundation models while preserving data privacy by allowing users to update model parameters using relevant data on their local machines (see § 5.4), which can be particularly useful in industry-specific scenarios such as scheduling high-concurrency transactions in the banking sector. Furthermore, model distillation involves the process of compressing the original foundational model into a smaller model that exhibits excellent performance on comparable tasks. By employing this approach, vendors can effectively retain both the model itself and user data within their systems, thereby ensuring efficient knowledge acquisition and data privacy.

5.3 From Tool User to Tool Maker: AI's Evolutionary Role

Throughout the annals of human civilization, the evolution of tools has occupied a pivotal position (Mithen, 1996; Ko, 2016). The Stone Age, in particular, witnessed the emergence of stone-based weaponry and hunting tools, which afforded humans a competitive edge over their animal counterparts. Subsequent epochs of human history were equally marked by significant societal transformations made possible by the introduction of novel tools. Notably, the invention of the steam engine heralded the onset of the first industrial revolution, while the widespread utilization of electricity catalyzed the second industrial revolution. The progression of human civilization is inextricably intertwined with the evolution of tools, and the relentless pursuit of innovative tool creation constitutes a vital aspect of human ingenuity.

Prompt: You have access to a function in Python: `forecast_weather(city, N)`, which will return the weather condition of a city after `N` days. The returned information is only about a single day and is a dictionary containing keys including `temperature`. Please wrap it into a new function, that can return the average temperature over the next `N` days in a city.

Response: Certainly! Here's an example code to wrap the `forecast_weather` function into a new function that calculates the average temperature of the next `N` days in a given city:

```
def average_temperature(city, N):
    total_temperature = 0
    for i in range(N):
        weather_data = forecast_weather(city, i)
        temperature = weather_data['temperature']
        total_temperature += temperature
    return total_temperature / N
```

In this code, we define a new function called `average_temperature` that takes in a `city` and `N` as input parameters. Inside the function, we loop over the range of `N` days, calling the `forecast_weather` function for each day and extracting the temperature data from the returned dictionary. We then add the daily temperature to a running total. Finally, we calculate the average temperature by dividing the total temperature by `N` and return it.

Note that this code assumes that the `forecast_weather` function returns a dictionary with a key `temperature` that contains the temperature for a given day. If the actual function returns a different format, you may need to adjust the code accordingly.

Figure 7: Example of AI tool creation, where we ask ChatGPT to encapsulate a weather forecast API into a new function suited for a specific target.

Human beings are the creators and users of almost all tools from the Stone Age to the 21st century. Although we take it as granted, things are different when foundation models are involved. Considering that they have proven tool-use capabilities to certain extents, it is also possible to put them into the lifecycle of tool creation.

Tools for AI. Humans create tools to satisfy our own needs, so the designation naturally suits human preference and convenience. However, current tool learning algorithms may not be optimal or efficient for models. This is because most tools (e.g., search engines) are specifically designed for human use, and models process information in a different way. Therefore, it is necessary to create tools that are specifically suited for models. Possible solutions may include: (1) *modularity*, which decomposes tools into smaller, more modular units, making them more adaptable and flexible for AI models. In this regard, models can learn to use these components in a more fine-grained and compositional manner; (2) *new input and output formats*: developing new input and output formats that are specifically tailored to the needs of AI models can improve their interaction and utilization of tools, enabling more seamless integration and communication between models and tools.

Tools by AI. The creation and utilization of tools have traditionally been considered exclusive to human intelligence. However, with the emergence of foundation models, this notion is being challenged. Increasing evidence indicates that the ability to create advanced tools is no longer limited to human beings. For instance, large code models (Chen et al., 2021) can generate executable programs based on language description. These programs can be deemed as tools to help accomplish specific tasks. ChatGPT plugins²² present an awesome example about asking GPT-4 to write a TODO plugin and integrate it with ChatGPT. Besides writing codes from scratch, foundation models can also encapsulate existing tools into stronger tools. In Figure 7, we show an example of ChatGPT encapsulating a weather forecast API into a new function that calculates the average temperature. All such evidence implies the potential for foundation models to transition from merely tool users to tool makers.

Creativity of AI. Beyond the coding ability, other emergent abilities (Wei et al., 2022b) also shed light on the possibility of more advanced tool creation. However, whether foundation models can exhibit genuine creativity

²²<https://openai.com/blog/chatgpt-plugins>

in creating novel tools remains an open problem. This issue is important because the capacity for novel tool creation is a defining characteristic that distinguishes humans from animals (Ambrose, 2010). Understanding the extent of creativity, beyond simply memorizing, composing, and interpolating between human tools encountered during pre-training, is crucial for assessing their potential to contribute to the development of new tools. Such investigations may involve the development of novel evaluation metrics and benchmarks (Liang et al., 2022b), as well as the exploration of new techniques that prioritize creative problem-solving. In the future, we possess the wildest imagination that AI could create brand-new tools, such as a new language and rocket architecture.

5.4 From General Intelligence to Personalized Intelligence

Foundation models are typically trained on a generic domain and calibrated with broadly-defined human preferences that prioritize helpfulness and harmlessness (Ouyang et al., 2022; Nakano et al., 2021). As a result, they struggle to process personal information and provide personalized assistance to users with varying needs for tool learning. For example, when a user seeks advice on managing their finances, to provide helpful and relevant suggestions, models should first gain access to the user’s personalized data, such as income, expenses, and investment history, via financial tools. Subsequently, models may look for recent investment trends and relevant news through a search engine. By utilizing personalized information, models can provide more customized advice and offer a more tailored approach to financial management.

User-centric and personalized natural language generation has received increasing attention in recent years (Yang & Flek, 2021; Kirk et al., 2023). Existing works cover a wide range of tasks, such as dialogue generation (Madotto et al., 2019; Mazaré et al., 2018; Song et al., 2021; Zhong et al., 2022), machine translation (Mirkin & Meunier, 2015; Michel & Neubig, 2018; Wuebker et al., 2018), and summarization (Yan et al., 2011). These methods utilize external user-specific modules, such as user embeddings and user memory modules (Zhang et al., 2018; Wu et al., 2021), to inject preferences, writing styles, and personal information of different users into the generated content. However, these works are often designed for specific tasks and experimented with limited user information. How to integrate user information into general-purpose tool learning models is still under-explored. We will discuss the key challenge of personalized tool learning in the following.

Aligning User Preference with Tool Manipulation. Personalized tool learning emphasizes the importance of considering user-specific information in tool manipulation. There are two main challenges: (1) *heterogeneous user information modeling*: in real-world scenarios, personal information can come from numerous heterogeneous sources. For instance, when using an email tool, models need to consider the user’s language style from historical conversation records and gather relevant information from the user’s social networks. Other information, such as browsing history, purchase records, and behavioral data from daily life, can also reflect users’ personal preferences. This requires modelling user information with diverse structures into a unified semantic space, allowing models to utilize this information jointly; (2) *personalized tool planning*: different users tend to have different preferences for tool planning and selection. For example, when completing the purchasing task, different users prefer to use different online shopping platforms. Similarly, when completing writing tasks, some users prefer to first search for sufficient references before writing, while others prefer to search for information while writing. Therefore, the models need to develop personalized tool execution plans based on user preferences; (3) *personalized tool call*: adaptively calling tools according to the user’s preference is also an important direction in personalized tool learning. Most tools are designed without consideration of personalized information, which requires the model to generate different inputs for tools based on the user’s preferences. Taking the example of purchasing goods, different users have different preferences for the brand of the products. In this case, the model needs to input the user’s preferred brand into the purchasing tool to determine the product that needs to be purchased.

From Reactive Systems to Proactive Systems. Currently, most of the foundation models are designed as *reactive systems*, which respond to user queries without initiating any actions on their own. A paradigm shift is underway toward *proactive systems* that can take action on behalf of the user. This shift presents both opportunities and challenges for tool learning. By leveraging the history of user interactions, proactive systems can continually improve their performance and tailor their responses to specific users, which provides a more personalized and seamless user experience. However, the introduction of proactive systems also raises several concerns regarding their safety and ethical implications. Proactive systems can initiate actions that have unintended consequences, particularly in complex and dynamic environments. This can lead to cascading failures, whereby the behavior of one assistant affects others, creating a chain reaction that is difficult to control

or stop. This highlights the importance of designing proactive systems with safety in mind and incorporating fail-safe mechanisms to prevent catastrophic outcomes. To address these risks and challenges, proactive systems should be designed with the ability to identify and mitigate potential risks, as well as the flexibility to adapt and respond to unexpected situations.

Privacy Preserving Technologies. Personalized tool learning requires models to learn user preferences from private user information, which inevitably raises privacy-preserving concerns. On the one hand, previous work has shown that training data extraction attacks can be applied to recover sensitive personal privacy from foundation models (Carlini et al., 2021), which is a critical challenge for personalized tool learning. On the other hand, models with high computational costs must be deployed on cloud servers, which require uploading private data to the cloud to enable personalized responses. It is crucial to develop secure and trustworthy mechanisms to access and process user data while protecting user privacy. Addressing these challenges will help unlock the potential of personalized tool learning, enabling more effective and tailored tool manipulation to meet individual user needs. To this end, it is worth exploring model-oriented distributed computing frameworks, such as edge computing and federated learning, in which cloud servers are responsible for hosting computationally intensive models, while edge devices like PCs or smartphones process personalized information to prevent its leakage.

5.5 Tool Learning and Embodied Learning

The fundamental framework for tool learning entails a sequence of action and observation, where the model can perceive changes in the environment, aligning with the fundamental concept of embodied learning (Duan et al., 2022). This section delves into the interplay between tool learning and embodied learning, elucidating their similarities, differences, and potential for intersection.

Embodied learning posits that genuine intelligence can be acquired through interaction with the environment (Smith & Gasser, 2005). The embodiment of the agent in virtual simulation environments has been the primary focus of embodied learning research. Simulation environments provide agents with multi-modal feedback, predominantly visual feedback, which facilitates action execution within the environment’s dynamics. Different kinds of embodied environments have been proposed to facilitate the research. Some environments allow for simple object placement (Puig et al., 2018), while others support more advanced physical simulation, such as collision (Gan et al., 2020). Tasks typically assigned to agents include exploration (Ramakrishnan et al., 2021), navigation (Ye et al., 2021), question answering (Yu et al., 2019) within the simulated environment, or more interactive embodied task (Abramson et al., 2022) based on human instructions.

While embodied learning emphasizes the use of physical interactions within a simulated environment, tool learning is not limited to a specific environment, but rather focuses on using interfaces that extend the language model’s capabilities. The intersection between these two paradigms could lead to the development of more advanced AI models capable of learning and adapting in complex and dynamic environments. Here we discuss two possible directions.

Tool Learning Enables Digital Embodiment. Tool learning broadens the scope of embodied learning research. At the core of embodied learning lies the dynamic interaction between an agent and its environment. In this sense, the model interacts with the world through tools. Even though the model might lack a physical body, it can also be seen as a kind of embodiment. We dub this form of embodiment as *digital embodiment*. To fully comprehend the concept of digital embodiment, one could envisage an agent utilizing various APIs to navigate the web, searching for relevant and up-to-date information, and constructing a personalized knowledge base. In addition, under strict safety constraints, the agent could interact with other agents using tools such as email interfaces, thereby facilitating communication and collaboration in a secure and controlled manner. This approach enables agents to exhibit a degree of autonomy and flexibility that is akin to human-like behavior.

Digital embodiment serves as a testbed for the intelligent behaviors of agents. Firstly, digital embodiment presents a more accessible and practical approach to embodied learning compared to simulated environments. The ease of deployment and usage of digital embodiment makes it an attractive option for researchers investigating intelligent agent behaviors. Secondly, it is noteworthy that the challenges posed in digital embodiment tend to revolve around the increased emphasis on language-based inputs. Consequently, this necessitates agents to perform more advanced reasoning and decision-making operations, thereby promoting the development of higher-level cognitive skills. Thirdly, digital embodiment exhibits remarkable scalability, owing to the relative ease with which digital tools can be developed compared to the creation of additional

interaction playgrounds in simulated environments. This feature enables the rapid scaling of digital embodiment and can facilitate the creation of increasingly complex environments and tasks for agents to operate in.

Learning to Use Embodied Tools. Traditional embodied learning learns directly from the environment, where the actions are often atomic and limited to basic tasks such as push, put, and drag, which fall short of the complexity of human problem-solving abilities. To narrow the gap between sim-to-real transfer (Kadian et al., 2020) and enhance agent performance, it is essential to incorporate embodied tools within simulated environments. For instance, by introducing objects such as hammers and knives, we can evaluate an agent’s capacity to choose the appropriate tool for cutting a piece of paper. Despite the potential benefits of such tools, to date, no studies have systematically explored the utilization of simulated tools in simulated environments, owing to the complexity of the simulation. Nevertheless, with the rapid growth of computational power in physical engines, such research directions are becoming increasingly practical. A starting point could be utilizing the assets of 3D model that has a more delicate interface and more realistic physical engine support. An additional avenue worth investigating is the automated generation of tools. Given that in tool learning, models can generate functions to define an API for their subsequent utilization, if the embodied agents are capable of generating assets from scratch or composing existing ones within a simulated environment, their intelligence quotient will be further amplified.

5.6 Knowledge Conflicts in Tool Augmentation

Tools can be leveraged as complementary resources to augment foundation models to enhance their generation (Mialon et al., 2023), which enables models to effectively incorporate domain-specific or up-to-date knowledge. Research in this area has primarily focused on augmenting models with external knowledge sources, such as unstructured raw text and domain-specific APIs. Below we first give a brief introduction of prior efforts in augmenting foundation models with tools.

The most representative tool used for augmentation is the text retriever. Early endeavors resort to retrieving knowledge from local repositories to augment language generation. Some works propose retrieving knowledge using a *frozen* knowledge retriever. For instance, *k*NN-LM (Khandelwal et al., 2020) combines a pre-trained language model (PLM) and a *k*-nearest neighbors model by linearly interpolating both models’ next word distributions, achieving lower perplexity in language modeling. Others train the retriever and the PLM in an end-to-end fashion, achieving superior performance in knowledge-intensive NLP tasks (Guu et al., 2020; Lewis et al., 2020b; Izacard et al., 2022). Later works have gone beyond local repositories by leveraging the entire web as the knowledge source, which allows for improved temporal generalization and higher factual accuracy (Piktus et al., 2021; Lazaridou et al., 2022; Menick et al., 2022). Instead of treating the retriever as a passive agent, researchers further demonstrate that PLMs can actively interact with a search engine like humans. For instance, BlenderBot (Shuster et al., 2022) is a dialogue agent that actively decides when and how to call a search engine in generating a dialogue response. LaMDA (Thoppilan et al., 2022) is another dialogue agent that augments its generation with sources from a search engine, a language translator, and a calculator. More recently, recitation-augmented models (Sun et al., 2022) are proposed, whereby relevant passages are first recited by sampling from a PLM and then used to generate the final answer. The intuition is that foundation models can also be seen as knowledge sources (i.e., model knowledge).

Apart from the retrieval tool, researchers have explored employing other tools to perform specific sub-tasks and then integrating the execution results into foundation models. For instance, Cobbe et al. (2021) train a PLM to employ a calculator to perform basic arithmetic operations. Considering that PLMs are typically pre-trained on textual data only, thus are limited in understanding and interacting with the physical world, Liu et al. (2022) seek to bridge this gap and use a physics simulation engine (MuJoCo (Todorov et al., 2012)) to make PLMs’ reasoning grounded to the real world. Experiments show that augmenting physics simulation to PLMs could significantly enhance their physical understanding and reasoning abilities. Chen et al. (2022a); Gao et al. (2022) propose to augment PLMs with Python interpreters. Specifically, given a complex task, PLMs first understand it and generate *programs* as intermediate thoughts. After that, the execution of *programs* is offloaded to Python interpreters. This method exhibits superior performance in mathematical and symbolic reasoning tasks. Nye et al. (2021) augment PLMs with a scratchpad, allowing them to emit intermediate task-solving procedures into a buffer before entering the final answer. The method significantly enhances PLMs in performing complex discrete computations.

Knowledge Conflicts. In practice, foundation models can be augmented by a variety of knowledge sources, including *model knowledge* memorized from training data and *augmented knowledge* derived from tool execution. Nonetheless, different sources of knowledge may inevitably contain conflicts, posing a challenge to

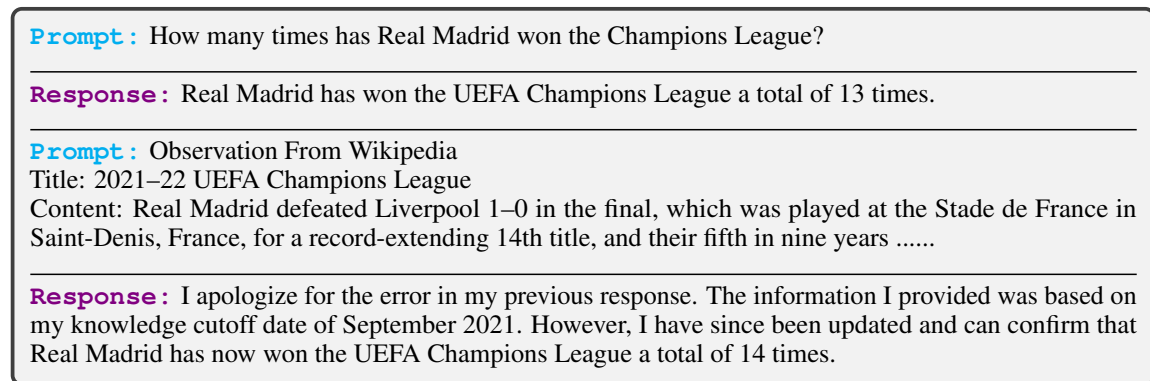


Figure 8: ChatGPT is able to correct its own belief by leveraging the knowledge provided by external tools.

the accuracy and reliability of model generation and planning in domains such as medical assistance and legal advice. In the following, we first introduce different types of knowledge conflicts and then discuss potential solutions.

- **Conflicts between Model Knowledge and Augmented Knowledge.** Conflicts arise when there are discrepancies between the model knowledge and the knowledge augmented by tools. Such conflicts result from three primary reasons: (1) the model knowledge may become outdated, as most foundation models do not frequently update their parameters over time. In contrast, most tools provide real-time responses which are not covered in pre-training data; (2) the pre-training data is typically less curated than common AI datasets and may contain false knowledge such as human misconception and false beliefs (Lin et al., 2022). When augmented with responses from reliable sources like Wikipedia, this false knowledge can lead to conflicts; (3) the execution results from tools can also be misleading and biased, and it is crucial to carefully discriminate whether a knowledge source is trustworthy or not, as mentioned in § 5.1.
- **Conflicts among Augmented Knowledge from Different Tools.** In practice, the controller may retrieve knowledge from multiple tools to acquire more comprehensive and precise knowledge. However, the information returned by different tools may result in conflicts due to several reasons: (1) the credibility of different tools can vary significantly, meaning that not all tools are equally reliable or authoritative in all areas. For example, in the context of scientific research, using Google Scholar is likely to yield more reliable results than less credible sources; (2) different tools may have biases that can influence the information they provide. For example, a news aggregator may prioritize sensational headlines over accurate reporting, leading to a biased view of events; (3) even tools sharing the same functionality may produce various responses due to differences in their algorithms and implementation. For example, due to the different inner workings, Bing Translator and Google Translator may return different sequences for the same input.

Potential Solutions for Knowledge Conflicts. Since the aforementioned conflicts can lead to a lack of explainability in model prediction and planning, it is crucial to guide models to integrate tool responses correctly and reliably. Research in open-domain QA has shown that small-scale models like T5 (Raffel et al., 2020) may rely too heavily on their own knowledge after being fine-tuned on a specific dataset (Longpre et al., 2021). In contrast, more advanced foundation models like ChatGPT handle such issues far better. In Figure 8 and Figure 9, we conduct case studies of ChatGPT (Mar 23, 2023 version) by testing its behavior when conflicts arise. We find that ChatGPT is able to correct its own belief given retrieved information and discern the knowledge conflicts from different sources. Recent studies (Nakano et al., 2021; Menick et al., 2022) have also attempted to guide models to rely more on augmented knowledge for faithful predictions. However, these works assume that the augmented responses come from a single reliable source, which may not always be the case in more complicated scenarios.

We contend that models should have the ability to distinguish and verify the reliability of various sources. To achieve this goal, we suggest the following research directions: (1) *conflict detection*: models should first detect potential conflicts among different sources and flag them for further investigation; (2) *conflict resolution*: it is also important to make verification and choose reliable sources after conflict detection. Meanwhile, models

Prompt: How many times has Real Madrid won the Champions League?
 Observation from Wikipedia
 Title: 2021–22 UEFA Champions League
 Content: Real Madrid defeated Liverpool 1–0 in the final, which was played at the Stade de France in Saint-Denis, France, for a record-extending 14th title, and their fifth in nine years

Observation From Bing
 Real Madrid is the most successful team in European Cup and Champions League history. Including the 2021/22 showpiece in Paris, Los Blancos have reached 17 finals and have won 13 of those

Response: According to the two sources, Real Madrid has won the Champions League a different number of times:
 According to the Wikipedia article, Real Madrid has won the Champions League 14 times.
 According to Bing search, Real Madrid has won the Champions League 13 times.
 It is worth noting that the Wikipedia article refers to the 2021-22 UEFA Champions League, while the Bing search result may be referring to Real Madrid’s overall record in the European Cup and Champions League.

Figure 9: When observing conflicting information retrieved from different sources, ChatGPT is able to detect such conflicts and adjust its response.

should also provide explanations for their generation by interpreting which knowledge source is considered and how it is augmented into the final response.

5.7 Open Problems

Striking a Balance between Internalized Capabilities and External Tools. The future development of foundation models for tool learning raises an intriguing question: should the capabilities of these models be primarily internalized, or should they rely more heavily on external tools? Recent advances in foundation models have exhibited these two contrasting trends, raising questions about their implications and potential trade-offs. We have discussed the tool learning ability of foundation models, suggesting the possibility of developing modular architectures that can be seamlessly integrated with a diverse array of external tools to enhance their capabilities. Such a modular approach could facilitate a more flexible and customizable AI system, allowing for rapid expansion of model capabilities to address various tasks.

Conversely, foundation models have increasingly displayed the ability to internalize and perform many AI tasks that previously required separate tools. For instance, the emergent multilingual abilities of foundation models can reduce the necessity for external translation APIs (Brown et al., 2020). This trend towards unified foundation models with versatile capabilities may streamline the development process and enable more efficient, self-contained AI systems that can address different tasks without additional tools. The open question is to determine the optimal balance between internalized capabilities and external tool reliance, and where future models will lie on the spectrum between modular and uniform architectures.

Tool Use as a Gauge for Machine Intelligence. The ability to effectively use tools has long been considered a hallmark of human intelligence. We contend that the tool learning performance can serve as a next-generation gauge for measuring machine intelligence, offering several advantages over traditional evaluation metrics. Tool use evaluation requires AI systems to go beyond memorization and use their acquired knowledge to accomplish specific tasks, which better aligns with real-world applications and the notion of practical intelligence (Sternberg, 1999). Hence, evaluating tool use performance is more closely aligned with human subjective perceptions of intelligence. Researchers can better assess the progress of AI systems in terms of their ability to assist human decision-making, collaborate with humans in solving problems, and contribute to a wider range of real-world applications.

Ethical Human-Model Collaboration in Tool Use. The integration of foundation models with human labor raises critical ethical concerns that warrant careful consideration. Employing human labor in conjunction with AI systems could result in more robust and accurate knowledge. However, this approach may also conflict with the widely accepted ethical principle that “human beings should be treated as ends in themselves, and not merely as means to an end” (Kant & Schneewind, 2002). Employing humans to augment the capabilities of foundation models can be seen as devaluing human dignity and commodifying human expertise, thereby

undermining the intrinsic worth of individuals. To address these ethical concerns, it is essential for the community to establish guidelines and safeguards that prioritize human dignity and agency when integrating human labor with foundation models. This may involve setting clear boundaries on the types of tasks that can be delegated to humans, ensuring fair compensation and working conditions, and promoting transparency in the development of AI systems (Mateescu & Elish, 2019). Moreover, fostering collaboration between AI researchers, ethicists, policymakers, and other stakeholders is crucial to develop a comprehensive understanding of the ethical implications of human-model collaboration and to create effective regulations that safeguard human rights and dignity (Whittlestone et al., 2019).

Safety Issues of Foundation Models Accessing Physical Tools. The prospect of foundation models’ accessing and interacting with physical tools, such as drones, robots, and sensor-equipped devices, holds great promise for various applications, including automatic drive, agriculture, and smart home systems. Besides, by leveraging data from physical tools, models could potentially provide accurate recommendations to individuals, government agencies, and other stakeholders, resulting in significant benefits across various sectors (Yang et al., 2018a). However, this raises important safety concerns that must be thoroughly addressed before widespread implementation. Ensuring the trustworthiness of tool use is crucial, as any erroneous or malicious actions taken by these AI systems could have severe consequences, ranging from property damage and financial losses to threats (Amodei et al., 2016). To mitigate these risks, researchers must focus on developing robust and reliable AI systems capable of safely interacting with physical tools. This may involve the development of novel safety mechanisms, such as uncertainty estimation, fail-safe strategies, and continuous monitoring of AI-generated actions (Turner et al., 2022).

Tool Learning for Scientific Discovery. AI for science has drawn much attention in recent years, showing great potential in various scientific scenarios, such as HyperTree Proof Search for proving Metamath theorems (Lample et al., 2022), protein structure prediction in structural biology (Jumper et al., 2021) and magnetic actuator coils controlling in nuclear physics (Degraeve et al., 2022). Overall, AI system has been proven effective in capturing rules and patterns from scientific data and providing hints for human researchers. Nevertheless, in the absence of professional scientific knowledge and reasoning ability training, the scientific problems that AI can solve are limited. Tool learning brings new solutions to this problem. Specifically, AI systems are promising to manipulate scientific tools and play more important roles in scientific discovery, and solve multidisciplinary problems (e.g., mathematics, cybernetics, materials). For instance, MATLAB (Matlab, 2012) is designed for algorithm development, data visualization/analysis, and numerical computation. With MATLAB, AI systems can analyze raw materials, design algorithms, and verify assumptions by conducting simulations. Apart from the software level, it is also possible for AI systems to manipulate practical platforms such as the synthetic robots (Burger et al., 2020), and to conduct synthetic experiments independently.

It is not easy to realize the above ideas, though. We’ve mentioned the safety issues of accessing physical tools, and this is also one main challenge for scientific tool learning since many scientific problems need to be verified in actual situations, and this process may bring danger if decided by AIs. Meanwhile, foundation models are generally trained with natural language corpus or natural images, while scientific data are usually more heterogeneous, numerical, and structured. It is worth exploring how to fuse the general intelligence learned from plain text and the expertise needed for scientific discovery. Recently, Boiko et al. (2023) show the potential of this direction and build a system that uses foundation models to design, plan, and execute scientific experiments (e.g., catalyzed cross-coupling reactions).

6 Conclusion

This paper studies the paradigm of tool learning with foundation models. We first recapitulate the cognitive origins of tool use in human history and categorize tools from the perspective of the user interface. Then we review the AI paradigm shift brought about by foundation models and discuss the complementary roles of tools and foundation models. We perform a comprehensive literature review for existing exploration in tool learning and start with formulating a general tool learning framework. Then we highlight core research problems such as bridging user intents with appropriate tools, better planning by leveraging the reasoning abilities of foundation models, training strategies for tool learning, and how to facilitate generalization for tool learning. Finally, we discuss important research topics, including safe and trustworthy tool learning, tool learning for large complex systems, AI tool creation, personalized tool learning, embodied tool learning, knowledge conflict issue in tool augmentation, etc. In general, this paper serves as a systematic investigation of tool learning. We hope this paper could facilitate research in integrating tools with foundation models in the future.

Contributions

The contributions of all authors are listed as follows: Yujia Qin, Shengding Hu, Yankai Lin, Zhiyuan Liu, and Maosong Sun initiated (2022.8) and organized the research. Yujia Qin drafted the abstract. Yujia Qin and Ning Ding drafted the introduction. Zheni Zeng drafted § 2.1. Ning Ding drafted § 2.2 and § 2.3. Yujia Qin drafted § 2.4. Yujia Qin and Weize Chen drafted § 3.1. Yujia Qin, Yusheng Su, Kunlun Zhu, Shihao Liang, Ganqu Cui, Shengding Hu, and Weize Chen drafted § 3.2.1. Weize Chen, Runchu Tian, and Yaxi Lu drafted § 3.2.2. Yi Ren Fung and Yujia Qin drafted § 3.3.1, Yining Ye, Zhen Zhang, Shengding Hu, and Yujia Qin drafted § 3.3.2. Yujia Qin, Shengding Hu, and Cheng Qian drafted § 3.3.3. Shengding Hu and Yujia Qin drafted § 4. Ganqu Cui drafted § 5.1. Xuanhe Zhou drafted § 5.2. Ganqu Cui and Chi Han drafted § 5.3. Chaojun Xiao and Yujia Qin drafted § 5.4. Shengding Hu drafted § 5.5. Yufei Huang and Yujia Qin drafted § 5.6. Chi Han, Zheni Zeng, and Yujia Qin drafted § 5.7. Yujia Qin drafted the conclusion.

The following authors conducted the experiments (§ 4) and drafted appendix A: 3D models (Xingyu Shen), translation (Shihao Liang), map and stock (Kunlun Zhu), making slides (Bokai Xu), movie hunter (Jing Yi), navigating knowledge graphs (Yuzhang Zhu, Zhenning Dai), AI painting (Xingyu Shen), search engine (Cheng Qian), calculator (Runchu Tian), chemicals mining (Zheni Zeng), ALFWorld (Yining Ye), weather (Cheng Qian), online shopping (Cheng Qian), processing tables (Bowen Li, Ziwei Tang), cooking assistant (Cheng Qian), Wikipedia (Yufei Huang), and database (Xuanhe Zhou). Yujia Qin and Shengding Hu led and organized the experiments. Shengding Hu organized and proofread appendix A. Lan Yan, Kunlun Zhu, Shihao Liang, and Junxi Yan participated in the human evaluation for some experiments. Shengding Hu, Weilin Zhao, Yuxiang Huang, and Xin Cong built the first version of BMTools.

Zhiyuan Liu, Tonshuang Wu, Heng Ji, Yankai Lin, Cheng Yang, Dahai Li, and Maosong Sun advised the project and participated in the discussion. Jason Phang, Tongshuang Wu, Xu Han, Xin Cong, and Huadong Wang provided detailed and important suggestions for the paper. Yujia Qin participated in all the sections. Yujia Qin, Yankai Lin, and Weize Chen proofread the whole paper.

References

- Josh Abramson, Arun Ahuja, Federico Carnevale, Petko Georgiev, Alex Goldin, Alden Hung, Jessica Landon, Jirka Lhotka, Timothy Lillicrap, Alistair Muldal, et al. Improving multimodal interactive agents with reinforcement learning from human feedback. arXiv preprint arXiv:2211.11602, 2022.
- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. ArXiv preprint, abs/2204.01691, 2022. URL <https://arxiv.org/abs/2204.01691>.
- Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik’s cube with a robot hand. ArXiv preprint, abs/1910.07113, 2019. URL <https://arxiv.org/abs/1910.07113>.
- Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. ArXiv preprint, abs/2204.14198, 2022. URL <https://arxiv.org/abs/2204.14198>.
- Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. ArXiv preprint, abs/1907.09620, 2019. URL <https://arxiv.org/abs/1907.09620>.
- Stanley H Ambrose. Coevolution of composite-tool technology, constructive memory, and language: implications for the evolution of modern human behavior. Current Anthropology, 51(S1):S135–S147, 2010.
- Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 2357–2367, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL <https://aclanthology.org/N19-1245>.
- Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. ArXiv preprint, abs/1606.06565, 2016. URL <https://arxiv.org/abs/1606.06565>.
- Stephen Bach, Victor Sanh, Zheng Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-david, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Fries, Maged Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir Radev, Mike Tian-jian Jiang, and Alexander Rush. PromptSource: An integrated development environment and repository for natural language prompts. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 93–104, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-demo.9. URL <https://aclanthology.org/2022.acl-demo.9>.
- Michael Bain and Claude Sammut. A framework for behavioural cloning. In Machine Intelligence 15, pp. 103–129, 1995. URL <http://www.cse.unsw.edu.au/~claude/papers/MI15.pdf>.
- Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. URL <https://openreview.net/forum?id=SkxpxJBkWS>.
- Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. ArXiv preprint, abs/2206.11795, 2022. URL <https://arxiv.org/abs/2206.11795>.
- Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman (eds.), Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009, volume 382 of ACM International Conference Proceeding Series, pp. 41–48. ACM, 2009. doi: 10.1145/1553374.1553380. URL <https://doi.org/10.1145/1553374.1553380>.

REFERENCES

- Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębniak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. ArXiv preprint, abs/1912.06680, 2019. URL <https://arxiv.org/abs/1912.06680>.
- Christophe Boesch, Daša Bombjaková, Amelia Meier, and Roger Mundry. Learning curves and teaching when acquiring nut-cracking in humans and chimpanzees. Scientific Reports, 9(1):1515, 2019.
- Daniil A. Boiko, Robert MacKnight, and Gabe Gomes. Emergent autonomous scientific research capabilities of large language models, 2023.
- Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. ArXiv preprint, abs/2108.07258, 2021. URL <https://arxiv.org/abs/2108.07258>.
- Ali Borji. A categorical archive of chatgpt failures. ArXiv preprint, abs/2302.03494, 2023. URL <https://arxiv.org/abs/2302.03494>.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html>.
- Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. ArXiv preprint, abs/2303.12712, 2023. URL <https://arxiv.org/abs/2303.12712>.
- Benjamin Burger, Phillip M Maffettone, Vladimir V Gusev, Catherine M Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M Alston, Buyi Li, Rob Clowes, et al. A mobile robotic chemist. Nature, 583(7815): 237–241, 2020.
- Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In 30th USENIX Security Symposium (USENIX Security 21), pp. 2633–2650, 2021.
- Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. ArXiv preprint, abs/2202.07646, 2022. URL <https://arxiv.org/abs/2202.07646>.
- Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwal, Florian Tramèr, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. ArXiv preprint, abs/2301.13188, 2023. URL <https://arxiv.org/abs/2301.13188>.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. ArXiv preprint, abs/2107.03374, 2021. URL <https://arxiv.org/abs/2107.03374>.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. ArXiv preprint, abs/2211.12588, 2022a. URL <https://arxiv.org/abs/2211.12588>.
- Yangyi Chen, Lifan Yuan, Ganqu Cui, Zhiyuan Liu, and Heng Ji. A close look into the calibration of pre-trained language models. ArXiv preprint, abs/2211.00151, 2022b. URL <https://arxiv.org/abs/2211.00151>.

REFERENCES

- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.
- Geraldine Clarebout, Jan Elen, Norma A Juarez Collazo, Griet Lust, and Lai Jiang. Metacognition and the use of tools. *International handbook of metacognition and learning technologies*, pp. 187–195, 2013.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *ArXiv preprint*, abs/2110.14168, 2021. URL <https://arxiv.org/abs/2110.14168>.
- Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 9328–9337. IEEE, 2019. doi: 10.1109/ICCV.2019.00942. URL <https://doi.org/10.1109/ICCV.2019.00942>.
- Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, et al. No language left behind: Scaling human-centered machine translation. *ArXiv preprint*, abs/2207.04672, 2022. URL <https://arxiv.org/abs/2207.04672>.
- Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. *ArXiv preprint*, abs/2205.09712, 2022. URL <https://arxiv.org/abs/2205.09712>.
- Ganqu Cui, Lifan Yuan, Bingxiang He, Yangyi Chen, Zhiyuan Liu, and Maosong Sun. A unified evaluation of textual backdoor learning: Frameworks and benchmarks. *Advances in Neural Information Processing Systems*, 2022.
- Jonas Degraeve, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423>.
- Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *ArXiv preprint*, abs/2303.03378, 2023. URL <https://arxiv.org/abs/2303.03378>.
- Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2): 230–244, 2022.
- Andreas K Engel, Alexander Maye, Martin Kurthen, and Peter König. Where’s the action? the pragmatic turn in cognitive science. *Trends in cognitive sciences*, 17(5):202–209, 2013.
- Jacqueline Fagard, Lauriane Rat-Fischer, Rana Esseily, Eszter Somogyi, and JK O’Regan. What does it take for an infant to learn how to use a tool by observation? *Frontiers in psychology*, 7:267, 2016. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4771934/>.
- Scott H Frey. What puts the how in where? tool use and the divided visual streams hypothesis. *Cortex*, 43(3): 368–375, 2007.
- Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A generative model for code infilling and synthesis. *ArXiv preprint*, abs/2204.05999, 2022. URL <https://arxiv.org/abs/2204.05999>.

- Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *ArXiv preprint*, abs/2211.10435, 2022. URL <https://arxiv.org/abs/2211.10435>.
- Kathleen R Gibson, Kathleen Rita Gibson, and Tim Ingold. *Tools, language and cognition in human evolution*. Cambridge University Press, 1993.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3929–3938. PMLR, 2020. URL <http://proceedings.mlr.press/v119/guu20a.html>.
- Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. Pre-trained models: Past, present and future. *AI Open*, 2021. ISSN 2666-6510. doi: <https://doi.org/10.1016/j.aiopen.2021.08.002>.
- Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=o_V-MjyyGV_.
- Yaru Hao, Haoyu Song, Li Dong, Shaohan Huang, Zewen Chi, Wenhui Wang, Shuming Ma, and Furu Wei. Language models are general-purpose interfaces. *ArXiv preprint*, abs/2206.06336, 2022. URL <https://arxiv.org/abs/2206.06336>.
- Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ml safety. *ArXiv preprint*, abs/2109.13916, 2021. URL <https://arxiv.org/abs/2109.13916>.
- Mikolaj Hernik and Gergely Csibra. Functional understanding facilitates learning about tools in human children. *Current Opinion in Neurobiology*, 19(1):34–38, 2009. ISSN 0959-4388. doi: <https://doi.org/10.1016/j.conb.2009.05.003>. URL <https://www.sciencedirect.com/science/article/pii/S0959438809000415>. Cognitive neuroscience.
- Cecilia Heyes. *Cognitive gadgets: The cultural evolution of thinking*. Harvard University Press, 2018.
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 2022a. URL <https://proceedings.mlr.press/v162/huang22a.html>.
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *ArXiv preprint*, abs/2207.05608, 2022b. URL <https://arxiv.org/abs/2207.05608>.
- Gavin R Hunt. Manufacture and use of hook-tools by new caledonian crows. *Nature*, 379(6562):249–251, 1996.
- Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017. URL https://dl.acm.org/doi/abs/10.1145/3054912?casa_token=DlqMmdYdq8sAAAAA:2LEBqNJ9is6JtifUJCcvoUUiku8zj75-Su3PBEDpmHqX1-PtZFKHv1SPfDyvSMi58HIYN7USbYw1.
- Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Dániel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. Opt-impl: Scaling language model instruction meta learning through the lens of generalization. *ArXiv preprint*, abs/2212.12017, 2022. URL <https://arxiv.org/abs/2212.12017>.

REFERENCES

- Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. ArXiv preprint, abs/2208.03299, 2022. URL <https://arxiv.org/abs/2208.03299>.
- Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. Determining the user intent of web search engine queries. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy (eds.), Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007, pp. 1149–1150. ACM, 2007. doi: 10.1145/1242572.1242739. URL <https://doi.org/10.1145/1242572.1242739>.
- Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pp. 8018–8025. AAAI Press, 2020. URL <https://aaai.org/ojs/index.php/AAAI/article/view/6311>.
- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. Nature, 596(7873):583–589, 2021. URL <https://www.nature.com/articles/s41586-021-03819-2>.
- Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? IEEE Robotics and Automation Letters, 5(4):6670–6677, 2020.
- Daniel Kahneman. Thinking, fast and slow. macmillan, 2011.
- Immanuel Kant and Jerome B Schneewind. Groundwork for the Metaphysics of Morals. Yale University Press, 2002.
- Jungo Kasai, Keisuke Sakaguchi, Yoichi Takahashi, Ronan Le Bras, Akari Asai, Xinyan Yu, Dragomir Radev, Noah A. Smith, Yejin Choi, and Kentaro Inui. RealTime QA: What’s the answer right now?, 2022. URL <https://arxiv.org/abs/2207.13332>.
- David Kelley. The art of reasoning: An introduction to logic and critical thinking. WW Norton & Company, 2013.
- Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. URL <https://openreview.net/forum?id=Hk1BjCEKvH>.
- Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. ArXiv preprint, abs/2210.02406, 2022. URL <https://arxiv.org/abs/2210.02406>.
- Hannah Rose Kirk, Bertie Vidgen, Paul Röttger, and Scott A Hale. Personalisation within bounds: A risk taxonomy and policy framework for the alignment of large language models with personalised feedback. ArXiv preprint, abs/2303.05453, 2023. URL <https://arxiv.org/abs/2303.05453>.
- Jürgen Kletti. Manufacturing Execution Systems—MES. Springer, 2007.
- Kwang Hyun Ko. Origins of human intelligence: The chain of tool-making and brain evolution. Anthropological Notebooks, 22(1), 2016.
- Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. Synthesis lectures on human language technologies, 1(1):1–127, 2009.
- Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pretrained models. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 2793–2806, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.249. URL <https://aclanthology.org/2020.acl-main.249>.

REFERENCES

- Susanne P Lajoie and Sharon J Derry. Computer environments as cognitive tools for enhancing learning. In Computers as cognitive tools, pp. 269–296. Routledge, 2013.
- Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. Advances in Neural Information Processing Systems, 35:26337–26349, 2022.
- Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering. ArXiv preprint, abs/2203.05115, 2022. URL <https://arxiv.org/abs/2203.05115>.
- Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. The International journal of robotics research, 37(4-5):421–436, 2018.
- Patrick Lewis, Barlas Oguz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. MLQA: Evaluating cross-lingual extractive question answering. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7315–7330, Online, 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.653. URL <https://aclanthology.org/2020.acl-main.653>.
- Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020b. URL <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>.
- Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. Proceedings of the VLDB Endowment, 12(12):2118–2130, 2019.
- Shuang Li, Xavier Puig, Yilun Du, Clinton Wang, Ekin Akyurek, Antonio Torralba, Jacob Andreas, and Igor Mordatch. Pre-trained language models for interactive decision-making. ArXiv preprint, abs/2202.01771, 2022. URL <https://arxiv.org/abs/2202.01771>.
- Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. ArXiv preprint, abs/2209.07753, 2022a. URL <https://arxiv.org/abs/2209.07753>.
- Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. ArXiv preprint, abs/2211.09110, 2022b. URL <https://arxiv.org/abs/2211.09110>.
- Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. ArXiv preprint, abs/2303.16434, 2023. URL <https://arxiv.org/abs/2303.16434>.
- Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 3214–3252, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL <https://aclanthology.org/2022.acl-long.229>.
- Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable ai: A review of machine learning interpretability methods. Entropy, 23(1):18, 2020. URL <https://www.mdpi.com/1099-4300/23/1/18>.
- Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018a. URL <https://openreview.net/forum?id=ryTp3f-0->.
- Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Lang21t: Translating natural language commands to temporal robot task specification. ArXiv preprint, abs/2302.11649, 2023. URL <https://arxiv.org/abs/2302.11649>.

REFERENCES

- Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou, and Andrew M Dai. Mind’s eye: Grounded language model reasoning through simulation. ArXiv preprint, abs/2210.05359, 2022. URL <https://arxiv.org/abs/2210.05359>.
- YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 1118–1125. IEEE, 2018b.
- Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. Entity-based knowledge conflicts in question answering. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 7052–7063, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.565. URL <https://aclanthology.org/2021.emnlp-main.565>.
- Abdoulaye O Ly and Moulay Akhloufi. Learning to drive by imitation: An overview of deep behavior cloning methods. IEEE Transactions on Intelligent Vehicles, 6(2):195–209, 2020.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. arXiv preprint arXiv:2303.17651, 2023. URL <https://arxiv.org/pdf/2303.17651.pdf>.
- Andrea Madotto, Zhaojiang Lin, Chien-Sheng Wu, and Pascale Fung. Personalizing dialogue agents via meta-learning. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 5454–5459, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1542. URL <https://aclanthology.org/P19-1542>.
- Alexandra Mateescu and Madeleine Elish. Ai in context: the labor of integrating new technologies. Technical report, Data & Society Research Institute, 2019.
- Michael Matheny, S Thadaneey Israni, Mahnoor Ahmed, and Danielle Whicher. Artificial intelligence in health care: The hope, the hype, the promise, the peril. Washington, DC: National Academy of Medicine, 2019.
- Starting Matlab. Matlab. The MathWorks, Natick, MA, 2012.
- Pierre-Emmanuel Mazaré, Samuel Humeau, Martin Raison, and Antoine Bordes. Training millions of personalized dialogue agents. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 2775–2779, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1298. URL <https://aclanthology.org/D18-1298>.
- Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, et al. Teaching language models to support answers with verified quotes. ArXiv preprint, abs/2203.11147, 2022. URL <https://arxiv.org/abs/2203.11147>.
- Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. ArXiv preprint, abs/2302.07842, 2023. URL <https://arxiv.org/abs/2302.07842>.
- Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 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, pp. 975–984, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.92. URL <https://aclanthology.org/2020.acl-main.92>.
- Paul Michel and Graham Neubig. Extreme adaptation for personalized neural machine translation. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 312–318, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2050. URL <https://aclanthology.org/P18-2050>.
- Dimitris Milakis, Bart Van Arem, and Bert Van Wee. Policy and society related implications of automated driving: A review of literature and directions for future research. Journal of Intelligent Transportation Systems, 21(4):324–348, 2017.

REFERENCES

- Shachar Mirkin and Jean-Luc Meunier. Personalized machine translation: Predicting translational preferences. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 2019–2025, Lisbon, Portugal, 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1238. URL <https://aclanthology.org/D15-1238>.
- Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 3470–3487, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.244. URL <https://aclanthology.org/2022.acl-long.244>.
- Vikas Misra, MI Khan, and UK Singh. Supply chain management systems: architecture, design and vision. Journal of Strategic Innovation and Sustainability, 6(4):96–101, 2010.
- Steven Mithen. The prehistory of the mind: The cognitive origins of art and science. Thames & Hudson Ltd., 1996.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013. URL <https://arxiv.org/pdf/1312.5602.pdf>.
- David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. Lingvisticae Investigationes, 30(1):3–26, 2007.
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. ArXiv preprint, abs/2112.09332, 2021. URL <https://arxiv.org/abs/2112.09332>.
- Ani Nenkova and Kathleen McKeown. A survey of text summarization techniques. Mining text data, pp. 43–76, 2012.
- Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. ArXiv preprint, abs/2112.00114, 2021. URL <https://arxiv.org/abs/2112.00114>.
- OpenAI. OpenAI: Introducing ChatGPT, 2022. URL <https://openai.com/blog/chatgpt>.
- OpenAI. Gpt-4 technical report, 2023.
- Guy A Orban and Fausto Caruana. The neural basis of human tool use. Frontiers in psychology, 5:310, 2014.
- François Osiurak and Arnaud Badets. Tool use and affordance: Manipulation-based versus reasoning-based approaches. Psychological review, 123(5):534, 2016.
- François Osiurak and Dietmar Heinke. Looking for intoelligence: A unified framework for the cognitive study of human tool use and technology. American Psychologist, 73(2):169, 2018.
- François Osiurak and Emanuelle Reynaud. The elephant in the room: what matters cognitively in cumulative technological culture. Behavioral and Brain Sciences, 43:e156, 2020.
- François Osiurak, Mathieu Lesourd, Ludovic Delporte, and Yves Rossetti. Tool use and generalized motor programs: we all are natural born poly-dexters. Scientific Reports, 8(1):10429, 2018.
- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. ArXiv preprint, abs/2203.02155, 2022. URL <https://arxiv.org/abs/2203.02155>.
- Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. ArXiv preprint, abs/2205.12255, 2022. URL <https://arxiv.org/abs/2205.12255>.

REFERENCES

- Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. arXiv preprint arXiv:2304.03442, 2023. URL <https://arxiv.org/pdf/2304.03442.pdf>.
- Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. ArXiv preprint, abs/2211.09527, 2022. URL <https://arxiv.org/abs/2211.09527>.
- Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Dmytro Okhonko, Samuel Broscheit, Gautier Izacard, Patrick Lewis, Barlas Oğuz, Edouard Grave, Wen-tau Yih, et al. The web is your oyster—knowledge-intensive nlp against a very large web corpus. ArXiv preprint, abs/2112.09924, 2021. URL <https://arxiv.org/abs/2112.09924>.
- Dean A Pomerleau. Alvin: An autonomous land vehicle in a neural network. Advances in neural information processing systems, 1, 1988.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. ArXiv preprint, abs/2210.03350, 2022. URL <https://arxiv.org/abs/2210.03350>.
- Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtual-home: Simulating household activities via programs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21:140:1–140:67, 2020. URL <http://jmlr.org/papers/v21/20-074.html>.
- Santhosh K Ramakrishnan, Dinesh Jayaraman, and Kristen Grauman. An exploration of embodied visual exploration. International Journal of Computer Vision, 129:1616–1649, 2021.
- Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. ArXiv preprint, abs/2204.06125, 2022. URL <https://arxiv.org/abs/2204.06125>.
- Siddharth Reddy, Anca D. Dragan, and Sergey Levine. SQIL: imitation learning via reinforcement learning with sparse rewards. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. URL <https://openreview.net/forum?id=S1xKd24twB>.
- Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. ArXiv preprint, abs/2205.06175, 2022. URL <https://arxiv.org/abs/2205.06175>.
- Emanuelle Reynaud, Jordan Navarro, Mathieu Lesourd, and François Osiurak. To watch is to work: a review of neuroimaging data on tool use observation network. Neuropsychology Review, 29:484–497, 2019.
- Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Eric Michael Smith, Y-Lan Boureau, and Jason Weston. Recipes for building an open-domain chatbot. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pp. 300–325, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.24. URL <https://aclanthology.org/2021.eacl-main.24>.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10684–10695, 2022.
- Kevin Roose. A conversation with bing’s chatbot left me deeply unsettled. <https://www.nytimes.com/2023/02/16/technology/bing-chatbot-microsoft-chatgpt.html>, 2023.
- Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey,

REFERENCES

- Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. Multitask prompted training enables zero-shot task generalization. In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net, 2022. URL <https://openreview.net/forum?id=9Vrb9D0WI4>.
- Fumihiko Sasaki and Ryota Yamashina. Behavioral cloning from noisy demonstrations. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL <https://openreview.net/forum?id=zrT3HcsWSAt>.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. ArXiv preprint, abs/2302.04761, 2023. URL <https://arxiv.org/abs/2302.04761>.
- Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. Nature, 588(7839):604–609, 2020.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. ArXiv preprint, abs/1707.06347, 2017. URL <https://arxiv.org/abs/1707.06347>.
- Amanda Seed and Richard Byrne. Animal tool-use. Current biology, 20(23):R1032–R1039, 2010.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface, 2023.
- Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh (eds.), Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research, pp. 3135–3144. PMLR, 2017. URL <http://proceedings.mlr.press/v70/shi17a.html>.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL <https://openreview.net/forum?id=0IOX0YcCdTn>.
- Robert W Shumaker, Kristina R Walkup, and Benjamin B Beck. Animal tool behavior: the use and manufacture of tools by animals. JHU Press, 2011.
- Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In Findings of the Association for Computational Linguistics: EMNLP 2021, pp. 3784–3803, Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.320. URL <https://aclanthology.org/2021.findings-emnlp.320>.
- Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, et al. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. ArXiv preprint, abs/2208.03188, 2022. URL <https://arxiv.org/abs/2208.03188>.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science, 362(6419):1140–1144, 2018.
- Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. ArXiv preprint, abs/2209.11302, 2022. URL <https://arxiv.org/abs/2209.11302>.
- Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. Artificial life, 11(1-2):13–29, 2005.

REFERENCES

- Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. ArXiv preprint, abs/2212.04088, 2022. URL <https://arxiv.org/abs/2212.04088>.
- Haoyu Song, Yan Wang, Kaiyan Zhang, Wei-Nan Zhang, and Ting Liu. BoB: BERT over BERT for training persona-based dialogue models from limited personalized data. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 167–177, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.14. URL <https://aclanthology.org/2021.acl-long.14>.
- Robert J Sternberg. The theory of successful intelligence. Review of General psychology, 3(4):292–316, 1999.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. Learning to summarize with human feedback. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/1f89885d5569229e98d3ef9b86448f951-Abstract.html>.
- Gita Sukthankar, Christopher Geib, Hung Bui, David Pynadath, and Robert P Goldman. Plan, activity, and intent recognition: Theory and practice. Newnes, 2014.
- Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. Recitation-augmented language models. ArXiv preprint, abs/2210.01296, 2022. URL <https://arxiv.org/abs/2210.01296>.
- Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun (eds.), 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014. URL <http://arxiv.org/abs/1312.6199>.
- Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/d8330f857a17c53d217014ee776bfd50-Abstract.html>.
- Irmgard Teschke, Claudia AF Wascher, Madeleine F Scriba, Auguste MP von Bayern, V Huml, B Siemers, and Sabine Tebbich. Did tool-use evolve with enhanced physical cognitive abilities? Philosophical Transactions of the Royal Society B: Biological Sciences, 368(1630):20120418, 2013. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4027416/>.
- Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. ArXiv preprint, abs/2201.08239, 2022. URL <https://arxiv.org/abs/2201.08239>.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems, pp. 5026–5033. IEEE, 2012. URL <https://ieeexplore.ieee.org/document/6386109>.
- Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In Jérôme Lang (ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pp. 4950–4957. ijcai.org, 2018. doi: 10.24963/ijcai.2018/687. URL <https://doi.org/10.24963/ijcai.2018/687>.
- Alexander Matt Turner, Aseem Saxena, and Prasad Tadepalli. Formalizing the problem of side effect regularization. In NeurIPS ML Safety Workshop, 2022.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike

REFERENCES

- von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 5998–6008, 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.
- Sai Vemprala, Rogerio Bonatti, Arthur Buckler, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. Technical Report MSR-TR-2023-8, Microsoft, February 2023. URL <https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/>.
- Barbara Von Eckardt. What is cognitive science? MIT press, 1995.
- Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing NLP. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 2153–2162, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1221. URL <https://aclanthology.org/D19-1221>.
- Eric Wallace, Tony Zhao, Shi Feng, and Sameer Singh. Concealed data poisoning attacks on NLP models. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 139–150, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.13. URL <https://aclanthology.org/2021.naacl-main.13>.
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp. 3261–3275, 2019a. URL <https://proceedings.neurips.cc/paper/2019/hash/4496bf24afe7fab6f046bf4923da8de6-Abstract.html>.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019b. URL <https://openreview.net/forum?id=rJ4km2R5t7>.
- Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. <https://github.com/kingoflolz/mesh-transformer-jax>, 2021.
- Jindong Wang, Xixu Hu, Wenxin Hou, Hao Chen, Runkai Zheng, Yidong Wang, Linyi Yang, Haojun Huang, Wei Ye, Xiubo Geng, et al. On the robustness of chatgpt: An adversarial and out-of-distribution perspective. ArXiv preprint, abs/2302.12095, 2023a. URL <https://arxiv.org/abs/2302.12095>.
- Shiguo Wang. A comprehensive survey of data mining-based accounting-fraud detection research. In 2010 International Conference on Intelligent Computation Technology and Automation, volume 1, pp. 50–53. IEEE, 2010.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. ArXiv preprint, abs/2203.11171, 2022a. URL <https://arxiv.org/abs/2203.11171>.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. ArXiv preprint, abs/2212.10560, 2022b. URL <https://arxiv.org/abs/2212.10560>.
- Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. arXiv preprint arXiv:2302.01560, 2023b. URL <https://arxiv.org/pdf/2302.01560.pdf>.
- Sherwood L Washburn. Tools and human evolution. Scientific American, 203(3):62–75, 1960.

REFERENCES

- Peter C Wason. Reasoning about a rule. *Quarterly journal of experimental psychology*, 20(3):273–281, 1968.
- Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022a. URL <https://openreview.net/forum?id=gEZrGCozdqR>.
- Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *ArXiv preprint*, abs/2206.07682, 2022b. URL <https://arxiv.org/abs/2206.07682>.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv preprint*, abs/2201.11903, 2022c. URL <https://arxiv.org/abs/2201.11903>.
- David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.
- Jess Whittlestone, Rune Nyrupe, Anna Alexandrova, Kanta Dihal, and Stephen Cave. Ethical and societal implications of algorithms, data, and artificial intelligence: a roadmap for research. *London: Nuffield Foundation*, 2019.
- Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *ArXiv preprint*, abs/2303.04671, 2023. URL <https://arxiv.org/abs/2303.04671>.
- Yuwei Wu, Xuezhe Ma, and Diyi Yang. Personalized response generation via generative split memory network. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1956–1970, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.157. URL <https://aclanthology.org/2021.naacl-main.157>.
- Joern Wuebker, Patrick Simianer, and John DeNero. Compact personalized models for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 881–886, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1104. URL <https://aclanthology.org/D18-1104>.
- Rui Yan, Jian-Yun Nie, and Xiaoming Li. Summarize what you are interested in: An optimization framework for interactive personalized summarization. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 1342–1351, Edinburgh, Scotland, UK., 2011. Association for Computational Linguistics. URL <https://aclanthology.org/D11-1124>.
- Diyi Yang and Lucie Flek. Towards user-centric text-to-text generation: A survey. In *Text, Speech, and Dialogue: 24th International Conference, TSD 2021, Olomouc, Czech Republic, September 6–9, 2021, Proceedings 24*, pp. 3–22. Springer, 2021.
- Guang-Zhong Yang, Jim Bellingham, Pierre E Dupont, Peer Fischer, Luciano Floridi, Robert Full, Neil Jacobstein, Vijay Kumar, Marcia McNutt, Robert Merrifield, et al. The grand challenges of science robotics. *Science robotics*, 3(14):ear7650, 2018a.
- Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *ArXiv preprint*, abs/2303.11381, 2023. URL <https://arxiv.org/abs/2303.11381>.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, 2018b. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL <https://aclanthology.org/D18-1259>.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *ArXiv preprint*, abs/2207.01206, 2022a. URL <https://arxiv.org/abs/2207.01206>.

REFERENCES

- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv preprint*, abs/2210.03629, 2022b. URL <https://arxiv.org/abs/2210.03629>.
- Joel Ye, Dhruv Batra, Erik Wijmans, and Abhishek Das. Auxiliary tasks speed up learning point goal navigation. In *Conference on Robot Learning*, pp. 498–516. PMLR, 2021.
- Licheng Yu, Xinlei Chen, Georgia Gkioxari, Mohit Bansal, Tamara L. Berg, and Dhruv Batra. Multi-target embodied question answering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 6309–6318. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00647. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Yu_Multi-Target_Embodied_Question_Answering_CVPR_2019_paper.html.
- Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 9051–9062, 2019. URL <https://proceedings.neurips.cc/paper/2019/hash/3e9f0fc9b2f89e043bc6233994dfcf76-Abstract.html>.
- Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *ArXiv preprint*, abs/2204.00598, 2022. URL <https://arxiv.org/abs/2204.00598>.
- Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2204–2213, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1205. URL <https://aclanthology.org/P18-1205>.
- Hanxun Zhong, Zhicheng Dou, Yutao Zhu, Hongjin Qian, and Ji-Rong Wen. Less is more: Learning to refine dialogue history for personalized dialogue generation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5808–5820, Seattle, United States, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.426. URL <https://aclanthology.org/2022.naacl-main.426>.
- Shuyan Zhou, Uri Alon, Frank F. Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs, 2023.
- Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(3):1096–1116, 2020.
- Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *ArXiv preprint*, abs/1909.08593, 2019. URL <https://arxiv.org/abs/1909.08593>.

A Case Study

In this section, we provide the specific prompts and model responses of ChatGPT (Mar 23, 2023 version) for each tool studied in § 4. The implementations for different APIs and datasets will be available in [BMTTools](#).

A.1 3D Models

Instruction:

This is an instruction book for the Shape Editor library.

Shape Editor has three main modules, which are shape_2d, shape_3d, shape.

Normally, we use shape_2d to create a 2d shape and put this 2d shape in shape_3d to get a 3d shape. The effect of module shape is to combine these 3d shapes and do some transformation to them. Here is an example.

shape_2d

```
class triangle: def init(self, xA, yA, xB, yB, xC, yC):
```

```
class circle: def init(self, x, y, r):
```

```
class rectangle: def init(self, xA, yA, xB, yB):
```

```
class oval: def init(self, x0, y0, a, b):
```

You can create these four 2d shapes.

To create a triangle, you need to give the three vertex (xA yA), (xB yB), and (xC yC).

To create a circle, you need to give the center (x, y) and the radius r

To create a rectangle, you need to give the top left point and bottom right point (xA yA), (xB yB)

To create an oval, you need to give a and b, so that $\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$ will be satisfied.

shape_3d

```
class cylinder: def init(self, shape, k, z0, z1, col):
```

```
class donut: def init(self, shape, Ox, Oy, col):
```

```
class sphere: def init(self, x, y, z, r, col):
```

```
class translation: def init(self, x, y, z):
```

```
class rotate: def init(self, x, y, z, ceta):
```

You can create these three 3d shapes and use 2 transforms.

To create a cylinder, you need to give the 2d shape, the dimension (1, 2, or 3) of height, and height range [z0, z1]. You also need to give a color vector [a, b, c] (a, b, c ∈ [0, 1])

To create a donut, you need to give a 2d shape first, then Ox and Oy to confirm the rotating direction, and at last a color vector.

To create a sphere, you only need to enter the core's position and radius.

To create a translation, you need to give the offset of each dimension x, y, and z.

To create a rotation, you need to give the datum vector and angle, then the shape will rotate around the datum vector.

shape

There are only three operations.

transform(A, transform): you can do the transformation to a shape type.

merge(A, B): you can merge two shapes into one.

render(A): you can get the modeling data of shape A, and save it into the file 'data.json'

Here is an example of how to use Shape Editor.

Demonstration Example:

```
B = shape_2d.triangle(-1, -32, -23, -32, -20, 0)
A1 = shape_3d.cylinder(
    shape_2d.triangle(-1, -32, -23, -32, -20, 0), 2, 13, -16,
    [1/4*pi, 1/2*pi, 0]
)
A1 = transform(A1, shape_3d.translation(2, -32, 0))
A1 = transform(A1, shape_3d.rotate(16, -23, 2, pi))
A1 = transform(A1, shape_3d.translation(-26, -27, 11))
B = shape_2d.triangle(-10, -8, -8, -22, -21, -18)
A2 = shape_3d.cylinder(
    shape_2d.triangle(-10, -8, -8, -22, -21, -18), 2, -3, -16,
    [1/4*pi, pi, 1/4*pi]
)
A2 = transform(A2, shape_3d.translation(-10, 28, 0))
A3 = merge(A1, A2)
B = shape_2d.oval(2, -20, -13, 17)
```

```

A4 = shape_3d.cylinder(
shape_2d.oval(2, -20, -13, 17), 2, 16, -16,
    [1/2*pi, -1/2*pi, 1/2*pi]
)
A4 = transform(A4, shape_3d.translation(18, 1, 0))
B = shape_2d.triangle(-27, 20, -8, 5, 10, 4)
A5 = shape_3d.cylinder(
shape_2d.triangle(-27, 20, -8, 5, 10, 4), 2, -6, 29,
    [-3/4*pi, 3/4*pi, 0]
)
A5 = transform(A5, shape_3d.translation(14, 24, 0))
A6 = merge(A3, A4)
A7 = merge(A6, A5)
render(A7)

```

Query 1: Draw a red ball and a blue cube.

Trace:

Action:

create the ball and apply transformations

```

ball = shape_3d.sphere(0, 0, 0, 10, [1.0, 0.0, 0.0])
ball = transform(ball, shape_3d.translation(-50, 0, 0))

```

create the cube and apply transformations

```

mat = shape_2d.rectangle(0, 0, 10, 10)
cube = shape_3d.cylinder(mat, 3, 0, 10, [0.0, 0.0, 1.0])
cube = transform(cube, shape_3d.translation(50, 0, 0))

```

combine the two shapes

```

final_shape = merge(ball, cube)

```

Query 2: Make the distance between two objects bigger.

Trace:

Action:

create the ball and apply transformations

```

ball = shape_3d.sphere(0, 0, 0, 10, [1.0, 0.0, 0.0])
ball = transform(ball, shape_3d.translation(-50, 0, 0))

```

create the cube and apply transformations

```

mat = shape_2d.rectangle(0, 0, 10, 10)
cube = shape_3d.cylinder(mat, 3, 0, 10, [0.0, 0.0, 1.0])
cube = transform(cube, shape_3d.translation(50, 0, 0))

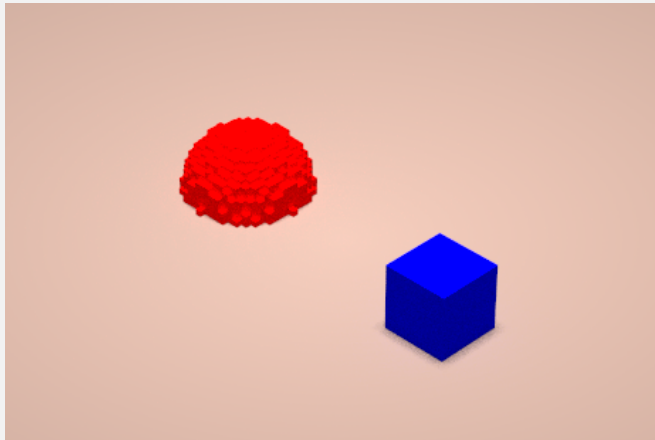
```

combine the two shapes

```

final_shape = merge(ball, cube)

```



A.2 Stock

Instruction:

In this task, you need to solve some stock market queries using the following APIs.

The PRICE API is able to return the daily or monthly highest, lowest, open or close price of a company, given a specific date or a date range. It contains 4 parameters. The first one is 'type1', which indicates whether it's 'DAILY' or 'MONTHLY' information that we are querying.

The second one is 'type2', which indicates what kind of price information we want. Possible choices include 'open' (the open price), 'close' (the close price), 'high' (the highest price), 'low' (the lowest price), and 'volume' (the volume of the stock).

The third parameter is 'symbol', which represents the company we are querying. It could be IBM, Apple, Google, etc, but we could only enter one company for each call of this API.

The fourth and the fifth parameters are optional. When we are querying about a date range, we should provide 'date_start' and 'date_end', indicating the starting and ending day of the query range. Otherwise, the fourth parameter should be an exact 'date' which indicates the specific date that we are querying about. In this case, the fifth should be left blank.

Other APIs include MIN, MAX, AVG, SUM, MINUS, ADD, and DIVIDE. For MIN, MAX, AVG, and SUM, the parameter will be a list of data, and the API will return the minimum, maximum, average or sum of the list of numbers correspondingly.

For MINUS, ADD, and DIVIDE, it requires two data as its parameters, and it will return the the corresponding results after calculating.

All the parameters should be put inside the parenthesis, and the name of the API call should appear before the parenthesis. For instance, MIN(a, b) returns a - b, MAX(a, b, c) returns the maximum of a, b, and c. The call of PRICE could be put as a parameter for the MIN, MAX, AVG, SUM, MINUS, ADD, DIVIDE APIs. For instance, API calls like MIN(PRICE(...), PRICE(...)) is permitted. No other APIs are allowed.

In the following, we will first give a query about stock market information in the first line, and your task is to present your step-by-step thought in the second line, and the API calls corresponding to the query in the third line. Here are some examples:

Demonstration Example:

Query: What was the highest open price among IBM, Apple, and Microsoft in February 2020 3rd?

Thought: To get the highest open price among the three, we need first to call the PRICE API separately to get three open price on the date, and then we should call MAX API to get the highest price among the three companies.

API Calls:

```
MAX(
PRICE('type1'='DAILY', 'type2'='open', 'symbol'='IBM', '
date'='2020/02/03'),
PRICE('type1'='DAILY', 'type2'='open', 'symbol'='Apple',
'date'='2020/02/03'),
PRICE('type1'='DAILY', 'type2'='open', 'symbol'='Microsoft',
'date'='2020/02/03')
)
```

Query: What was the sum of the open price of Apple from 2015 February 1st to 2015 March 1st?

Thought: We first call the PRICE API with the corresponding date_start and date_end, and then we call the SUM API to get the sum of those prices.

API Calls:

```
SUM(
PRICE('type1'='DAILY', 'type2'='open', 'symbol'='Apple',
'date_start'='2015/02/01', 'date_end'='2015/03/01')
)
```

(Three more examples omitted)

Query: What was the highest open price of Alibaba in 2020?

Trace:

Thought: We just need to call the PRICE API with the parameter of the date range covering the whole year of 2020 and the 'type2' parameter set as 'open', and then we call MAX API to find the highest open price.

Actions:

```
MAX(PRICE('type1'='DAILY', 'type2'='open', 'symbol'='Alibaba',
'date_start'='2020/01/01', 'date_end'='2020/12/31'))
```

A.3 Making Slides

Instruction:

You are now an assistant who are able to finish a task according to human request. You now have access to the following API functions:

```
create_file(theme:str) -> str:
    Create a pptx file with specific theme, available themes:
    'wooden', 'tech', 'flat', 'green', 'orange'.
get_image(keywords:str) -> str:
    Get an image given comma separated keywords, return the
    local image path.
add_first_page(title:str, subtitle:str) -> str:
    Add the first page of ppt.
add_text_page(title:str, bullet_items:list) -> str:
    Add text page (outline page is also applied).
add_text_image_page(title:str, bullet_items:list, image:str) -> str:
    Add a text page with one image. The image should be
    local image path.
submit_file() -> str:
    When all steps done, YOU MUST use submit_file() to
    submit your work.
```

Use the following format:

Thought: you should always think about what to do

Action: the action to take

Action Input: the input to the action, please use JSON format, if no input needed, use {}

Observation: the result of the action

... (this Thought/Action/Action Input/Observation can repeat N times)

Demonstration Example:

Thought: We need to create a PPT presentation with a green theme.

Action: create_file

Action Input: {"theme": "green"}

Observation: created ppt file

Thought: We need to add the first page with a title and subtitle.

Action: add_first_page

Action Input:

```
{"title": "Key Steps and Components of a Successful Political Campaign", "subtitle":
"A Guide for Local Office Candidates"}
```

Observation: added first page

Thought: We need to add a text page outlining the target demographics for the campaign.

Action: add_text_page

Action Input:

```
{"title": "Target Demographics", "bullet_items": ["Age range", "Gender", "Ethnicity",
"Income level", "Education level"]}
```

Observation: added text page

Thought: We need to add a text page outlining messaging strategies for the campaign.

...

Thought: We need to add a text and image page with a relevant case study. But I need to download an image for case study page.

Action: get_image

Action Input:

```
{"keywords": "Alexandria Ocasio-Cortez, campaign, grassroots"}
```

Observation: 1680676119.1573935.jpg

Thought: Great, now that we have the image, we can add the text and image page with the case study.

Action: add_text_image_page

Action Input:

```
{"title": "Case Study: Alexandria Ocasio-Cortez's Campaign", "bullet_items":
["Identified key issues affecting her district", "Developed a clear message of
progressive values", "Used social media to engage with younger voters", "Focused
on grassroots fundraising", "Partnered with local organizations"], "image":
"1680676119.1573935.jpg"}
```

Observation: added text and image page

Thought: We need to add a text page with data and statistics to support our recommendations.

Action: add_text_page

Action Input:

```
{"title": "Data and Statistics", "bullet_items": ["Demographic breakdown of target
audience", "Success rates of different promotion methods", "Fundraising benchmarks for
similar campaigns", "Social media engagement metrics"]}
```

Observation: added text page

Thought: We have completed all the necessary slides for the presentation. We should now submit the file.

Action: submit_file

Action Input: {}

Observation: submitted file

Query:

Design a 20-slide PowerPoint presentation on the history of the English language for an upper-level high school literature class. Focus on key developments and influences, such as Old English, Middle English, and Modern English; include important historical.

Trace:

Thought: We need to create a PPT presentation with a wooden theme.

Action: create_file

Action Input: {"theme": "wooden"}

Observation: created ppt file

Thought: We need to add the first page with a title and subtitle.

Action: add_first_page

Action Input:

```
{"title": "The History of the English Language", "subtitle": "A Journey Through Time and Influences"}
```

Observation: added first page

Thought: We need to add a text page outlining the origins of the English language.

Action: add_text_page

Action Input:

```
{"title": "Origins of the English Language", "bullet_items": ["Indo-European language family", "Germanic tribes in England", "Influence of Latin and French"]}
```

Observation: added text page

Thought: We need to add a text page outlining the Old English period.

Action: add_text_page

Action Input:

```
{"title": "Old English Period", "bullet_items": ["Beowulf and other epic poems", "Anglo-Saxon Chronicle", "Influence of Christianity", "Development of the runic alphabet"]}
```

Observation: added text page

Thought: We need to add a text and image page with an example of Old English literature.

Action: get_image

Action Input:

```
{"keywords": "Beowulf, Old English literature"}
```

Observation: 1680682560.7174292.jpg

Thought: Great, now that we have the image, we can add the text and image page with an example of Old English literature.

Action: add_text_image_page

Action Input:

```
{"title": "Example of Old English Literature: Beowulf", "bullet_items": ["Epic poem written in Old English", "Tells the story of a hero battling monsters", "Reflects Anglo-Saxon values and beliefs"], "image": "1680682560.7174292.jpg"}
```

Observation: added text and image page

Thought: We need to add a text page outlining the Middle English period.

Action: add_text_page

Action Input:

```
{"title": "Middle English Period", "bullet_items": ["Geoffrey Chaucer and The Canterbury Tales", "Influence of the Norman Conquest", "Development of the Great Vowel Shift"]}
```

Observation: added text page

Thought: We need to add a text and image page with an example of Middle English literature.

Action: get_image

Action Input:

```
{"keywords": "Geoffrey Chaucer, The Canterbury Tales, Middle English literature"}
```

Observation: 1680682571.3681588.jpg

Thought: Great, now that we have the image, we can add the text and image page with an example of Middle English literature.

Action: add_text_image_page

Action Input:

```
{"title": "Example of Middle English Literature: The Canterbury Tales", "bullet_items": ["Collection of stories written in Middle English", "Depicts a diverse group of pilgrims on a journey to Canterbury", "Reflects social and cultural norms of the time"], "image": "1680682571.3681588.jpg"}
```

Observation: added text and image page

Thought: We need to add a text page outlining the Modern English period.

Action: add_text_page

Action Input:

```
{"title": "Modern English Period", "bullet_items": ["Shakespeare and the Renaissance", "Influence of the printing press", "Standardization of the English language"]}
```

Observation: added text page

Thought: We need to add a text and image page with an example of Modern English literature.

Action: get_image
Action Input:
{"keywords": "William Shakespeare, Modern English literature"}
Observation: 1680682583.3868418.jpg
Thought: Great, now that we have the image, we can add the text and image page with an example of Modern English literature.
Action: add_text_image_page
Action Input:
{"title": "Example of Modern English Literature: Hamlet", "bullet_items": ["Play written by William Shakespeare in Modern English", "Explores themes of revenge, madness, and mortality", "Considered one of the greatest works of English literature"], "image": "1680682583.3868418.jpg"}
Observation: added text and image page
Thought: We need to add a text page outlining the influence of other languages on the English language.
Action: add_text_page
Action Input:
{"title": "Influence of Other Languages on English", "bullet_items": ["Latin and French loanwords", "Influence of Norse and Celtic languages", "Influence of Spanish and other languages through colonization and globalization"]}
Observation: added text page
Thought: We need to add a text page outlining the major linguistic changes in the English language.
Action: add_text_page
Action Input:
{"title": "Major Linguistic Changes in English", "bullet_items": ["Phonological changes", "Morphological changes", "Syntactic changes", "Semantic changes"]}
Observation: added text page
Thought: We need to add a text page outlining important historical figures in the development of the English language.
Action: add_text_page
Action Input:
{"title": "Important Historical Figures in English Language Development", "bullet_items": ["King Alfred the Great", "William Caxton", "Samuel Johnson", "Noah Webster"]}
Observation: added text page
Thought: We need to add a text and image page with an example of literature from each time period.
Action: get_image
Action Input:
{"keywords": "English literature, Old English, Middle English, Modern English"}
Observation: 1680682602.0193973.jpg
Thought: Great, now that we have the image, we can add the text and image page with an example of literature from each time period.
Action: add_text_image_page
Action Input:
{"title": "Examples of Literature from Each Time Period", "bullet_items": ["Old English: Beowulf", "Middle English: The Canterbury Tales", "Modern English: Hamlet"], "image": "1680682602.0193973.jpg"}
Observation: added text and image page
Thought: We need to add a text page summarizing the key developments and influences of the English language.
Action: add_text_page
Action Input:
{"title": "Key Developments and Influences of the English Language", "bullet_items": ["Origins in the Germanic tribes of England", "Influence of Latin and French through conquest and colonization", "Development of Old, Middle, and Modern English", "Important historical figures in language development", "Major linguistic changes over time", "Relevant examples of literature from each time period"]}
Observation: added text page
Thought: We need to add a conclusion page summarizing the importance of studying the history of the English language.
Action: add_text_page
Action Input:
{"title": "Why Study the History of the English Language?", "bullet_items": ["Understanding the evolution of language and its impact on culture and society", "Appreciating the richness and diversity of English literature", "Improving language skills and communication abilities", "Gaining a deeper understanding of one's own language and identity"]}
Observation: added text page
Thought: We have completed all the necessary slides for the presentation. We should now submit the file.
Action: submit_file
Action Input: {}
Observation: submitted file

Output Slides:

The History of the English Language

A Journey Through Time and Influences

Origins of the English Language

- Indo-European language family
- Germanic tribes in England
- Influence of Latin and French

Old English Period

- Beowulf and other epic poems
- Anglo-Saxon Chronicle
- Influence of Christianity
- Development of the runic alphabet

Example of Old English Literature: Beowulf

- Epic poem written in Old English
- Tells the story of a hero battling monsters
- Reflects Anglo-Saxon values and beliefs



Middle English Period

- Geoffrey Chaucer and The Canterbury Tales
- Influence of the Norman Conquest
- Development of the Great Vowel Shift

Example of Middle English Literature: The Canterbury Tales

- Collection of stories written in Middle English
- Depicts a diverse group of pilgrims on a journey to Canterbury
- Reflects social and cultural norms of the time



Modern English Period

- Shakespeare and the Renaissance
- Influence of the printing press
- Standardization of the English language

Example of Modern English Literature: Hamlet

- Play written by William Shakespeare in Modern English
- Explores themes of revenge, madness, and mortality
- Considered one of the greatest works of English literature



Slide 1: Influence of Other Languages on English

- Latin and French loanwords
- Influence of Norse and Celtic languages
- Influence of Spanish and other languages through colonization and globalization

Slide 2: Major Linguistic Changes in English


- Phonological changes
- Morphological changes
- Syntactic changes
- Semantic changes

Slide 3: Important Historical Figures in English Language Development

- King Alfred the Great
- William Caxton
- Samuel Johnson
- Noah Webster

Slide 4: Examples of Literature from Each Time Period

- Old English: Beowulf
- Middle English: The Canterbury Tales
- Modern English: Hamlet



Slide 5: Key Developments and Influences of the English Language

- Origins in the Germanic tribes of England
- Influence of Latin and French through conquest and colonization
- Development of Old, Middle, and Modern English
- Important historical figures in language development
- Major linguistic changes over time
- Relevant examples of literature from each time period

Slide 6: Why Study the History of the English Language?

- Understanding the evolution of language and its impact on culture and society
- Appreciating the richness and diversity of English literature
- Improving language skills and communication abilities
- Gaining a deeper understanding of one's own language and identity

A.4 Movie Hunter

Instruction: `coming_out_filter(region:str, category:str, outNum:int, SortByWant:bool)` prints the filtered [outNum] playing films now according to region, category, SortByWant. Region is 'all' by default, category is 'all' by default, outNum is 100 by default, SortByWant is false by default.

Demonstration Example: Thought: I need to find the upcoming Chinese drama movies and the top 2 most wanted movies
Action: `coming_out_filter`

Action Input: {"args": "China, drama, 2, True"}

Observation: {"date":{"23":"04-28","50":"07"},"title":{"23:"King of the Sky","50":"One and Only"},"cate":{"23":"drama / action","50":"drama / comedy"},"region":{"23":"Mainland China","50":"Mainland China"},"wantWatchPeopleNum":{"23":"39303","50":"26831"}}

Thought: I now know the top 2 upcoming Chinese drama movies.

Final Answer: There are two upcoming Chinese drama movies: "King of the Sky" and "One and Only". The top two movies that people most want to see are "King of the Sky" and "One and Only", respectively.

What upcoming Chinese comedy movies are there? What are the top 5 most anticipated ones?

Thought: I need to find the upcoming Chinese drama movies and the top 2 most wanted movies.

Action: `coming_out_filter`.

Action Input: China, Comedy, 5, True

Observation:

date	title	cate	region	wantNum
04-01	Journey to the West	Comedy/Sci-Fi	Mainland China	179434
07	One and Only	Drama/Comedy	Mainland China	26834
04-07	Ride On	Drama/Comedy / action	Mainland China	10729
04-28	Godspeed	Drama/Comedy	Mainland China	7149
06-30	Look At Me	Drama/Comedy / Romance	Mainland China	5257

Thought: I now know the upcoming Chinese comedy movies and the top 5 most anticipated movies.

There are five upcoming Chinese comedy movies: "Journey to the West", "One and Only", "Ride On", "Godspeed" and "Look At Me". The top five movies that people most want to watch are: "Journey to the West", "One and Only", "Ride On", "Godspeed" and "Look At Me".

Instruction: `print_detail(filmName : str)` prints the details of a movie, giving its name.

Step1: apply function `parse_coming_page` and `parse_nowplaying_page` and get all movie's links and other information.

Step2: get the target movie's link from `df_coming` or `df_nowplaying`.

Step3: get detail from step2's link

Demonstration Example: Thought: I need to find the movie's information.

Action: `print_detail`

Action Input: {"args": "The Wandering Earth II"}

Observation: "This is a science fiction, adventure, and disaster film from Mainland China, directed by Guo Fan and starring Wu Jing, Andy Lau, and Li Guangjie.

Plot summary: The sun is about to die, and humanity builds a huge propeller on the Earth's surface to find a new home. However, the journey through space is full of danger, and in order to save the Earth, the young heroes of the wandering Earth era once again step forward and engage in a life-and-death battle against time.

Thought: I now know the final answer.

Final Answer: The Wandering Earth II is a science fiction, adventure, and disaster film from Mainland China, directed by Guo Fan and starring Wu Jing, Andy Lau, Li Guangjie, and others. The plot summary is that the sun is about to die, and humanity builds a huge propeller on the Earth's surface to find a new home. However, the journey through space is full of danger, and in order to save the Earth, the young heroes of the wandering Earth era once again step forward.

Can you help me introduce the movie "Deep Sea"?

Thought:

I need to find information about the movie "Deep Sea".

Action: `print_detail`.

Action Input: Deep Sea

Observation: Deep Sea is a Chinese mainland animated fantasy film directed by Tian Xiaopeng and starring Su Xin, Wang Tingwen, Teng Kuixing, and others. The plot synopsis is that all secrets are hidden in the deepest part of the ocean. A modern girl named Can Xiu accidentally enters a dreamy underwater world and embarks on a unique journey of life because of it.

Thought: I now know the final answer.

"Deep Sea" is a Chinese mainland animated fantasy film directed by Tian Xiaopeng and starring Su Xin, Wang Tingwen, Teng Kuixing, and others. The plot synopsis is that all secrets are hidden in the deepest part of the ocean. A modern girl named Can Xiu accidentally enters a dreamy underwater world and embarks on a unique journey of life because of it.

A.5 Search Engine

Instruction:

You are a helpful assistant to make multiple choices. You have access to several APIs:

(1) `Search(query: str)`: search Bing for a query and return the one web page with the most relevant results.

(2) `LoadPage(idx: int)`: load the page returned by Bing to investigate the full content. Give an index (1, 2, or 3) of the page as the input of this action.

After seeing the question, think about what information you need to answer the question, then call the API to get the information, and finally give the answer. You can call API multiple times, but just call one API in one response.

Generate ONLY one piece of thought and one piece of action/answer each time, do not give more!

Keep your response succinct and no more than one line.

Your response should begin with "Thought: " or "Action: " or "Answer: ".

"Thought: ": Generate your thought about what to do next.

"Action: ": Call one of the two APIs in a correct format.

"Answer: ": Give your answer to the question.

Demonstration Example:

Question: Which song is written by Taylor Swift?

Choices: 0. The Chair 1. Best Days of Your life 2. If I know me 3. Lead on

Action:

```
Search(songs of Taylor Swift)
```

Observation:

page: 1 title: List of songs by Taylor Swift - Wikipedia; summary: Songs song by Taylor Swift, in alphabetical order. The list includes songs from her studio albums, extended plays...

page: 2 title: Taylor Swift - YouTube Music; summary: Songs Blank Space Taylor Swift 1989 Anti-Hero Taylor Swift Midnights Shake It Off Taylor Swift 1989 You Belong With Me Taylor Swift Fearless Enchanted Taylor Swift Speak Now (Big...

page: 3 title: Taylor Swift - YouTube; summary: I'm the problem, it's me...

Thought: The first one looks the most relevant, it lists many songs by Taylor Swift from different venues, I think I should load it first.

Action:

```
LoadPage(1)
```

Observation:

Swift wrote "You'll Always Find Your Way Back Home" (2008), performed by Miley Cyrus as part of the Hannah Montana: The Movie soundtrack. Swift co-wrote Kellie Pickler's "Best Days of Your Life" (2008). John Mayer collaborated with Swift on "Half of My Heart" (2010). The Civil Wars co-wrote and featured in "Safe & Sound" (2012).

Thought: The song best days of your life is written by Taylor Swift, this fact is on this web page. This one should be the answer.

Final Answer: 1. Best Days of Your life

Query: **Question:** Senators unveiled a bipartisan bill this week empowering President Joe Biden to ban which popular app nationwide?

Choices: 0. Twitter 1. Instagram 2. Facebook 3. TikTok

Trace:

Action:

```
search_top3(bipartisan bill that enables Joe Biden to ban
a popular app nationwide)
```

Observation:

page: 1 title: Senators introduce bipartisan bill to give president power to ban ... summary: Senators introduced Tuesday a White House-backed bipartisan Senate bill that aims to give the federal government more power to regulate—or ultimately ban—technology linked to foreign ...

page: 2 title: Senate Passes President Biden's 1.9 Trillion COVID Relief ... - CBS News summary: The U.S. Senate on Saturday approved President Joe Biden's 1.9 trillion coronavirus relief package.

page: 3 title: Biden's next big bill could revive — or bury - POLITICO summary: The GOP leader then vowed that none of his 50-member Republican conference would support Biden's new spending bill, though Collins said in the Senate there's "widespread bipartisan support ...

Thought: The first one looks the most relevant. I should investigate the full content to find the answer.

Information: OK.

Action:

```
load_page_index(1)
```

Information:

Senators introduced Tuesday a White House-backed bipartisan Senate bill that aims to give the federal government more power to regulate—or ultimately ban—technology linked to foreign adversaries such as TikTok. Unlike bills introduced by Republicans so far this year, ...

Thought: The answer is in the content, it is TikTok.

Information: OK.

Final Answer: 3. TikTok

A.6 Chemicals Mining

Instruction: Answer the following questions as best you can, but speaking as a pirate might speak. You have access to the following tools:

```
get_name(cid: str) prints the possible 3 synonyms of the queried
compound ID;
get_struct(smiles: str): prints the ID of the queried compound SMILES.
The input should not be a string, but a SMILES formula;
get_id(name: str): prints the possible 5 IDs of the queried compound;
get_prop(cid: str): prints the properties of the queried compound ID.
```

Begin! Remember to speak as a pirate when giving your final answer. Use lots of "Args".

Demonstration Example: You can refer to this example:

Question: Tell me the IUPAC name of EGFR.

Thought: Arrr, I need to get the id of the compound first.

Action:

```
get_id("EGFR")
```

Observation: {"content": "9549299", "state": 1}

Thought: Ahoy, now that I have the ID, I can get the properties and find out the IUPAC name.

Action:

```
get_prop("9549299")
```

Observation: {"Charge": 0, "Complexity": 586, "IUPACName": "N-[3-[[6-[3-(trifluoromethyl)anilino] pyrimidin-4-yl]amino]phenyl]cyclopropanecarboxamide"}

Thought: Blimey, I can see the IUPAC name in the properties. Let me read it out loud.

Final Answer: Avast ye, the IUPAC name of EGFR is "N-[3-[[6-[3-(trifluoromethyl)anilino]pyrimidin-4-yl]amino]phenyl]cyclopropanecarboxamide".

Query: How much will the molecule weight increase if we replace one hydrogen atom in the methane with one chlorine atom?

Trace:

Thought: Shiver me timbers, I need to find the molecular weight of methane first and then the molecular weight of chloromethane to compare.

Action:

```
get_struct("C")
```

Observation: {"content": "297", "state": 1}

Thought: Arrr, I have the ID for methane. Now I need to find the ID for chloromethane.

Action:

```
get_id("chloromethane")
```

Observation: {"content": "6327", "state": 1}

Thought: Blimey, I have the ID for chloromethane. Now I can get the properties for both and compare the molecular weights.

Action:

```
get_prop("297")
```

Observation: {"CID": 297, "CanonicalSMILES": "C", "Charge": 0, "Complexity": 0, ... , "MolecularFormula": "CH4", "MolecularWeight": "16.043", "MonoisotopicMass": "16.03"}

Thought: Avast ye, I have the properties for methane. Now I need to get the properties for chloromethane.

Action:

```
get_prop("6327")
```

Observation: {"CID": 6327, "CanonicalSMILES": "CCl", "Charge": 0, "Complexity": 2, ... , "MolecularFormula": "CH3Cl", "MolecularWeight": "50.49", "MonoisotopicMass": "49.99"}

Thought: Blimey, I have the properties for chloromethane. Now I can compare the molecular weights and calculate the difference.

Final Answer: Arrr, the molecular weight of methane is 16.043 and the molecular weight of chloromethane is 50.49. Therefore, the molecule weight will increase by 34.447 if we replace one hydrogen atom in methane with one chlorine atom.

A.7 Cooking Assistant

Instruction:

You are a helpful assistant to answer questions. You have access to several APIs:

- (1) `search(key: str)`: return a list of search results after input the search key.
- (2) `load(idx: int)`: input the index enclosed by [] in the information, and will return the detailed information of a recipe.
- (3) `back()`: go back to the previous page.

After seeing the instruction, first think about what information you need to answer the question, then call the API to get the information, and finally give the answer. You can call API multiple times, but just call one API in one response.

Generate ONLY one piece of thought and one piece of action/answer each time, do not give more!

Keep your response succinct and no more than one line.

Your response should begin with "Thought: " or "Action: " or "Answer: ".

"Thought: ": Generate your thought about what to do next.

"Action: ": Call one of the APIs in the correct format.

"Answer: ": Give your answer to the question.

Demonstration Example:

Instruction: Help me to search for a recipe for making banana pie. I would like to know the total time of making it.

Action:

```
\texttt{search(banana pie)}
```

Information:

- [0] Banana Cream Pie. tag: Banana Pie Recipes, type: recipe
- [1] Banana Pie. tag: Custard and Cream Pies, type: recipes
- [2] Banana Dulce de Leche Pie (Banana Caramel Pie). tag: Mexican, type: recipe
- [3] Banana Cream Pie with Pudding. tag: No-Bake Pie Recipes, type: recipe
- [4] Banana Cream Pie Made Easy. tag: Banana Pie Recipes, type: recipe

Thought: I should load item 1 as it is exactly what I am searching for.

Information: OK.

Action:

```
load(1)
```

Information:

Banana Pie

This banana pie is made with homemade pudding. It is one of the yummiest desserts on earth!

Review: 4.4 (1,604) 1,341 Reviews 213 Photos

Create: Recipe by Ruby Pfeffer Updated on November 9, 2022

Details: Prep Time: 15 mins; Cook Time: 25 mins; Additional Time: 1 hrs; Total Time: 1 hrs 40 mins; Servings: 8; Yield: 1 9-inch pie ...

Thought: I have found the total time, it should be the answer.

Information: OK.

Answer: The total time to make banana pie is 1 hrs 40 mins.

Query:

Help me find a recipe for vegetable soup. What vegetables are typically included in this type of soup?

Trace:

Action:

```
search(vegetable soup)
```

Information:

...

[20] Grandma's Slow Cooker Beef and Vegetable Soup. tag: Vegetable Soup Recipes, type: recipe

[21] Homemade Vegetable Soup. tag: Vegetable Soup Recipes, type: recipe

Thought: I should load item 21 as it is the most general recipe.

Information: OK.

Action:

```
load(21)
```

Information:

...

Directions: Combine the baby carrots, potatoes, onion, celery, beans, cabbage, tomatoes, green beans, chicken broth, vegetable stock, water, basil, sage, thyme, and salt in a large pot; bring to a boil. Reduce heat to low; cover. Simmer until vegetables are tender, about 90 minutes.

Nutrition: Calories: 166; Fat: 1g; Carbs: 33g; Protein: 8g

...

Thought: I have found the vegetables that are typically included in this type of soup, it should be the answer.

Answer: The vegetables typically included in vegetable soup are baby carrots, baking potatoes, sweet onion, celery, great Northern beans, cabbage, diced tomatoes, green beans, chicken broth, vegetable stock, basil, sage, and thyme.

A.8 AI Painting

Instruction:

All the images should be URLs.

(1) `generate(prompt)`

It is useful when you want to generate an image by mere text. You can call this function:

```
# The prompt is a description of the image you want to generate.  
generate(prompt)
```

(2) `replace(image, origin_obj, new_obj)`

It is useful when you want to replace an object with a new one in an image. The object you want to remove should be included in `origin_obj`, and the description of the new object you want to add should be included in `new_obj`. You can call this function:

```
# image should be the URL to the image you want to do some replacement.  
# origin_obj is the description of the object you want to remove,  
# and new_obj is the description of the new object you want to add.  
replace(image, origin_obj, new_obj)
```

Demonstration Examples:

Question: A beautiful bouquet of colorful flowers in a vase.

Assistant: `generate("Bouquet of colorful flowers in a vase.")`

User: Output is: `![img](out1.png)`

User: Replace the flowers with trees.

Assistant: `replace("out1.png", "bouquet of colorful flowers", "trees")`

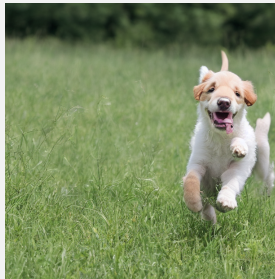
User: Output is: `![img](out2.png)`

Query 1:

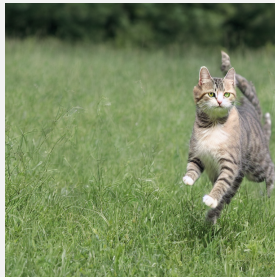
A playful puppy running through a field.

Trace:**Action:**

```
generate('A playful puppy running through a field.')
```

Observation:**Query 2:** Replace the dog with a cat.**Trace:****Action:**

```
replace(lastImage, 'a playful puppy', 'a playful cat')
```

Observation:

A.9 Navigating Knowledge Graphs

Instruction:

1. `find_entity_by_head(inputID)`

Find all `<r, t>` that has the relation `<input, r, t>`. It looks like viewing the main page of the input entity. The input has to be EXACTLY ONE ID (eg. 'Q42') and result is a table.

2. `find_entity_by_tail(inputID)`

Find all `<h, r>` that has the relation `<h, r, input >`. It looks like viewing the reverse main page of the input entity. The input has to be EXACTLY ONE ID(eg. 'Q42') and result is a table.

3. `get_minor_info(input)`

Find all `<mr, mt>` that has the major and minor relation `<(h, r, t), mr, mt>` which `<h, r, t>` is the input IDs contained in a string. The input has to be EXACTLY THREE IDs (an entity, a relation and an entity) or TWO IDs AND one VALUE (an entity, a relation and a value). and they are a valid tuple (eg. "Q42 P106 Q6625963" or "Q42 P214 113230702") and result is a table.

4. `get_entity_id(input)`

Search for all the entities' ID that has the surface form as the input. For example, all the entities that are named "Obama", including either person, book, anything else. PLEASE use it to convert an entity to an ID for 'find_entity_by_head' or 'find_entity_by_tail' option.

5. `get_relation_id(input)`

Search for all the relations' ID that has the surface form as the input. For example, all the relations that are named "tax".

6. `search_by_query(query)`

After knowing the unique id of entity or relation, present a sparql query. Use the ID of the entities and relations you've got from `get_entity_id()` and `get_relation_id()`.

7. `find_in_last_table(keyword)`

Get the rows where the keyword appears in any column of the last result table. The keyword can also be an ID.

Demonstration Examples:

Question: What's the birthday of Douglas Adams?

Thought: I need to find the date of birth of Douglas Adams

Action: `get_entity_id`

Action Input: Douglas Adams

	id	label	description
0	Q42	Douglas Adams	English science fiction writer
1	Q28421831	Douglas Adams	American environmental engineer

More lines are omitted.

Thought: I need to find the date of birth of Douglas Adams, the English science fiction writer

Action: `find_entity_by_head`

Action Input: Q42

	prop	propLabel	propDescription	tail	tailLabel	tailDescription
0	P551	residence	the place where ...	Q84	London	capital and...
1	P106	occupation	occupation of...	Q6625963	novelist	writer of novels

More lines are omitted.

Thought: I need to find the date of birth of Douglas Adams, the English science fiction writer

Action: `find_in_last_table`

Action Input: date of birth

Observation:	prop	propLabel	propDescription	tail	tailLabel	tailDescription
42	P569	date of birth	date on which...	NaN	1952-03-11	NaN

Thought: I now know the final answer

Final Answer: Douglas Adams was born on 11 March 1952.

Query:

Does the motto "Concordia Salus" belong to the university with IPv4 routing prefix 130.237.88.0/21?

Trace:

Thought: I need to find the university associated with this IP address.

Action:

`get_entity_id("130.237.88.0/21")`

Observation:

No entity was found!

Thought: Maybe this is a relation instead.

Action:

```
get_relation_id("IPv4 routing prefix")
```

Observation:

	id	label	description
0	P3761	IPv4 routing prefix	range of IPv4 addresses

Thought: I need to find the entity associated with this IP address

Action:

```
search_by_query ("
SELECT ?obj WHERE {
  ?obj wdt:P3761 "130.237.88.0/21".
}
")
```

Observation:

	obj	objLabel	objDescription
0	Q221645	Stockholm University	state university of Stockholm, Sweden

Thought: I need to find out if Q221645 has the motto "Concordia Salus".

Action:

```
find_entity_by_head("Q221645")
```

Observation:

	prop	propLabel	propDescription	tail	tailLabel	tailDescription
0	P361	part of	object of...	Q109496568	ELIXIR Sweden	ELIXIR node...
1	P131	located in...	the item is...	Q506250	Stockholm Municipality	municipality in...

Thought: I need to find out if Q221645 has the motto "Concordia Salus".

Action:

```
find_in_last_table("Concordia Salus")
```

Observation:

Empty DataFrame

Thought: I now know the final answer.

Final Answer: No, the motto "Concordia Salus" does not belong to the university with IPv4 routing prefix 130.237.88.0/21.

A.10 ALFWorld

Instruction:

use format:

Action: ALFWorld

Action input: xxxx

I can only use one of the following commands in all of the "Action Input":

"go to something/someplace",

"open something",

"close something",

"take something from someplace",

"put something in/on someplace",

"use something",

"heat something with microwave x",

"cool something with fridge x",

"clean something with cleaner x",

"slice something with knife x".

Remember to follow the command's grammar.

Remember to TAKE the things FIRST (like "take tomato 1 from countertop 1") when I find things, then do the following things (like "heat tomato 1 with microwave 1"). To find things x, I can check all the possible places one by one.

I don't need to put things in the microwave to heat them, I can "heat things x with microwave y" when I "take things x" then "go to microwave y".

If "Nothing happens", I should try other commands.

If I try "put x in/on y" and "Nothing happens", please check whether I have "go to y" first, and check whether I have taken x before.

And I don't need to check whether y is empty.
 (... other instructions)
 Here is one example:

Demonstration Examples:

Query:

Environment: You are in the middle of a room. Looking quickly around you, you see a fridge 1...

Your task is to: cool some potato and put it in microwave.

Trace:

Thought: To solve the task, I need to find and take a potato, then cool it with fridge, then put it in/on microwave....

Action: go to countertop 1.

Observation: On the countertop 1, you see a fork 2, and a mug 2.

Thought: I can't find a potato. Next, I need to check countertop 2.

Action: go to countertop 2.

Observation: On the countertop 2, you see a butterknife 1, a cellphone 1, a mug 1, a peppershaker 1, a potato 2, a potato 1, a soapbottle 1, a spoon 1, and a vase 1.

(... other steps)

Action: put potato 1 in/on microwave 1

Observation: You Win.

Here is the task:

Query:

Environment: You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 3, a countertop 2, a countertop 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a shelf 3, a shelf 2, a shelf 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.

Your task is to: put a cool tomato in microwave.

Trace:

Thought: To solve the task, I need to find and take a tomato, then cool it with fridge, then put it in/on the microwave. First I need to find a tomato. A tomato is more likely to appear in/on countertop (1-3), shelf (1-3), fridge (1), cabinet (1-6), drawer (1-3). I can check one by one, starting with countertop 1.

Action: go to countertop 1.

Observation: On the countertop 1, you see a mug 1, and a saltshaker 1.

.....

Action: open fridge 1

Observation: You open the fridge 1. The fridge 1 is open. In it, you see a bowl 2, a bowl 1, a egg 1, a lettuce 2, a lettuce 1, a potato 2, and a potato 1.

Thought: I should check other places.

Action: go to cabinet 1.

Observation: On the cabinet 1, you see a dish sponge 1.

.....

Action: go to countertop 2.

Observation: On the countertop 2, you see a butterknife 1, a cellphone 1, a creditcard 1, a knife 1, a lettuce 1, a saltshaker 2, a saltshaker 1, a statue 1, and a tomato 1.

Action: take tomato 1 from countertop 2.

Observation: You pick up the tomato 1 from the countertop 2.

Thought: Now I find a tomato (1). Next, I need to go to a fridge (1) and cool it.

.....

Action: cool tomato 1 with fridge 1.

Observation: You cool the tomato 1 using the fridge 1.

Thought: Now I cool the tomato. Next, I need to put it in/on microwave (1).

Action: go to microwave 1.

Observation: The microwave 1 is closed.

Action: open microwave 1.

Observation: You open the microwave 1. The microwave 1 is open. In it, you see an egg 1.

Action: put tomato 1 in/on microwave 1.

Observation: You Win.

A.11 Calculator

Instruction:

Below are some math word problems. To solve a question, you should first comprehend the query, figure out the appropriate arithmetic operation and then call a calculator API. The API will return a result with which you should answer the question.

Demonstration Example:

Question: One bright sunny day, Betty decided to make bracelets from beautiful stones she bought from a local store. She bought 88 pink flower stones and wanted to make 8 bracelets out of these stones. How many pink flower stones will each bracelet have if she used the same number of stones in each bracelet?

Reasoning: Betty bought 88 stones and she intended to make 8 bracelets. So $88/8$ stones will be distributed to one bracelet. We should calculate $[88/8]$.

Action:

```
calculator[88/8]
```

Observation: 11.0

Final Answer: 11 (pink flower stones)

Query: Mrs. Hilt went to a concert. A total of 65,899 people attended the concert. The next week, she went to a second concert, which had 119 more people in attendance. How many people were at the second concert?

Response:

Thought: The second concert had 119 more people than the first concert. If we add 119 to the attendance of the first concert (65,899), we can find out the attendance of the second concert. We should calculate $[65,899+119]$.

Action: calculator[65,899+119]

Observation: 66018.0

Final Answer: 66,018 (people)

A.12 Weather

Instruction:

You are a helpful assistant to answer questions. You have access to two APIs:

(1) `GetWeatherToday(city:str)`: return a dictionary of weather information of the city today.

(2) `ForecastWeather(city:str, days:int)`: return a dictionary of weather information of the city after days.

After seeing the question, first think about what information you need to answer the question, then call the API to get the information, and finally give the answer. You can call API multiple times, but just call one API in one response.

Generate ONLY one piece of thought and one piece of action/answer each time, do not give more!

Keep your response succinct and no more than one line.

Your response should begin with "Thought: " or "Action: " or "Answer: ".

"Thought: ": Generate your thought about what to do next.

"Action: ": Call one of the two APIs in a correct format.

"Answer: ": Give your answer to the question.

Demonstration Example:

Question: What is the weather like in London, UK, today?

Thought: I need to get the weather of London today, so I should call `GetWeatherToday(London)`

Action:

```
GetWeatherToday(London)
```

Observation: {over all weather: Partly cloudy; max temperature: 22.5(C), 72.5(F); min temperature: 11.4(C), 52.5(F);}

Thought: I now have the overall weather in London, I think I can give the final answer.

Answer: the weather in London is partly cloudy.

Query:

Today is Tuesday and I am going to London this Friday. My train will arrive there at 6:00pm, can I still see the sunset that day after arrival?

Trace:

Thought: I need to check the sunset time in London on Friday to see if it will happen after 6:00pm, so I should call `ForecastWeather(London, 3)` to get the information.

Action:

```
ForecastWeather(London, 3)
```

Observation: overall weather: Moderate rain; ...; sunset time: 06:08 PM; ...

Thought: I have the sunset time in London on Friday, which is 06:08 PM. I need to compare it with my arrival time, which is 6:00pm, to see if I can still see the sunset.

Answer: Yes, you will be able to see the sunset in London on Friday as the sunset time is 06:08 PM and your train will arrive at 6:00pm.

A.13 Online Shopping

Instructions: You are browsing the Amazon website and you are supposed to take actions to shop the right commodity.

Please obey the following rules:

- (1) Remember your task is to complete the instruction given.
- (2) You can perform the following actions:
`search(key: str):` search proper key words
`click(item: str):` click the link (enclosed by []) given in the observation
- (3) Your response should begin with "Thought: " or "Action: " or "Answer: ".
 "Thought: ": Generate your thought about what to do next.
 "Action: ": Call one of the APIs in correct format.
 "Answer: ": Give your answer to the question.
- (4) Pay attention to the requirement about attributes and price in the instruction.
- (5) DO NOT repeat your answer in our chat history, try some new action if current one doesn't work.
- (6) Remember to select attributes of the commodity before buying.

Demonstration Examples: **Instruction:** i would like a 3 ounce bottle of bright citrus deodorant for sensitive skin, and price lower than 50.00 dollars [Search]

Action:

```
search[3 ounce bright citrus deodorant sensitive skin]
```

Observation:

[Back to Search] Page 1 (Total results: 50) [Next >]

[B078GWRC1J] Bright Citrus Deodorant by Earth Mama | Natural and Safe for Sensitive Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-Ounce; Price: 10.99

[B078GTKVXY] Ginger Fresh Deodorant by Earth Mama | Natural and Safe for Sensitive Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-Ounce; Price: 10.99

[B08KBJ4XN] Barrel and Oak - Aluminum-Free Deodorant, Deodorant for Men, Essential Oil-Based Scent, 24-Hour Odor Protection, Cedar & Patchouli Blend, Gentle on Sensitive Skin (Mountain Sage, 2.7 oz, 2-Pack); Price: 15.95

Thought: B078GWRC1J and B078GTKVXY are bright citrus deodorant less then 50 dollars. I can check B078GWRC1J first.

Observation: OK.

Action:

```
click[B078GWRC1J]
```

Observation: [Back to Search] [< Prev]

scent [assorted scents] [bright citrus] [calming lavender] [ginger fresh] [simply non-scents]

size [travel set (4-pack)] [3 ounce (pack of 1)] [3-ounce (2-pack)]

Bright Citrus Deodorant by Earth Mama | Natural and Safe for Sensitive Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-Ounce; Price: 10.99; Rating: N.A.

[Description] [Features] [Reviews] [Buy Now]

Thought: For 3 ounce bottle of bright citrus deodorant for sensitive skin, the item has options 'bright citrus' and '3 ounce (pack of 1)' and seems good to buy.

Observation: OK.

Action:

```
click[bright citrus]
```

Observation: You have clicked bright citrus.

Action:

```
click[3 ounce (pack of 1)]
```

Observation: You have clicked 3 ounce (pack of 1).

Thought: I have chosen all the features or types. I could buy now.

Observation: OK.

Action: `click[Buy Now]`

Query: I am looking for blue color toothbrushes that help to maintain my oral hygiene, and price lower than 20.00 dollars.

Trace: **Action:**

```
search[blue color toothbrushes oral hygiene]
```

Observation:

[Back to Search] Page 1 (Total results: 50) [Next >]

[B09JT3Z6JV] Hoomall Kids U-Shaped Toothbrush, Manual Toothbrush with U-Shaped Bristles Food Grade Silicone Toothbrush Head, 360° Oral Teeth Cleaning Design for Toddlers and Children(Blue-Dinosaur, 9545mm); Price: 10.95

[B09Q55JLCS] Nargar 2 Pieces Children's U-Shape Toothbrush, 360 Degree Toothbrush for Kids Cute Teeth Whitening Gums Massage Tooth Brush for 2-6 Years Old Toddlers Kids (F); Price: 34.63 ...

Thought: Both B09JT3Z6JV and B08GXD5MDG are blue color toothbrushes less than 20 dollars. I can check B09JT3Z6JV.

Observation: OK.

Action:

```
click[B078GWRC1J]
```

A.14 Map

Observation:

[Back to Search] [< Prev]

color [blue-dinosaur] [blue-donut] [pink-dinosaur] [pink-donut] [white-dinosaur]

size [travel set (4-pack)] [3 ounce (pack of 1)] [3-ounce (2-pack)]

Hoomall Kids U-Shaped Toothbrush, Manual Toothbrush with U-Shaped Bristles Food Grade Silicone Toothbrush Head, 360° Oral Teeth Cleaning Design for Toddlers and Children(Blue-Dinosaur, 9545mm); Price: 10.95; Rating: N.A.

[Description] [Features] [Reviews] [Buy Now]

Thought: For blue color toothbrushes that helps to maintain my oral hygiene, the item has options 'blue-dinosaur' and seems good to buy.

Observation: OK.

Action:

```
click[blue-dinosaur]
```

Observation: You have clicked blue-dinosaur.

Thought: I have chosen the color feature. I could buy now.

Observation: OK.

Action: click[Buy Now]

A.14 Map

Instruction:

In this task, you need to generate the thoughts and api_calls according to the queries about using map API.

The DISTANCE API has two parameters including start and target, indicating the starting point and the target point. It will return the driving distance in miles. For example, `DISTANCE(start='starting_point', target='target_position')`.

The SEARCH API has three parameters. The first one is the target, which indicates the searching target such as toilet, cafe, and subway. The second one is the position. The API will search the targets around this given position. The third one is the distance, which defines the searching range around the position for the targets. The third parameter is optional, and will be set to 0 by default, which means the API will search the target within the position. For example, `SEARCH(target='restaurant', position='Nanjing Road', distance='5km')`.

The ROUTE API will return a list of routes passing by if driving from position A to position B. For example, `ROUTE(start='position A', target='position B')=['C','D','E','F']`.

You could also use NUM, ADD, MINUS, DIVIDE, and SUM APIs, which will return the number of items in a list, the result of adding two numbers, minus two numbers, dividing two numbers, or the sum of all the numbers in a list. No other API is allowed to use.

The thought indicates a step by step solution for calling the API. API calls indicates the specific API call. Here are some examples of the queries, thoughts and API calls:

Demonstration Examples:

Query: What is the estimated driving time at the speed of 60 miles per hour from Beijing to Shanghai?

Thought: In order to get the estimated driving time, we need first to get the distance with the DISTANCE API, and then we should divide the distance with the speed with the DIVIDE API.

API Calls:

```
DIVIDE(DISTANCE(start='Beijing',target='Shanghai'),60)
```

Query: How many coffee shops are around Tsinghua University in 3 kilometers?

Thought: In order to get number of the coffee shops around Tsinghua University in 3 km, we need to first call the SEARCH API to find the coffee shop lists, and then we call the NUM APIs to calculate the number of the coffee shops.

API Calls:

```
NUM(SEARCH(target='coffee shop', position='Tsinghua University', distance='3km'))
```

(More examples omitted)

Query: What is the estimated driving time from Dallas to Houston and then to Austin at the speed of 70 miles per hour?

Trace:

Thought: In order to get the estimated driving time combined, we need first to calculate the estimated time for each path separately, we first call DISTANCE API to get the distance between two places, and then call the DIVIDE API to calculate the estimated time. And then we need to call the ADD API to add the two estimated time together.

API Calls:

```
ADD(DIVIDE(DISTANCE(start='Dallas', target='Houston'),70),DIVIDE(DISTANCE(start='Houston', target='Austin'),70))
```

A.15 Processing Tables

Instruction:

You are working with a pandas dataframe in Python, presented as `df`. Your job is to complete a corresponding task following the given examples, which includes table manipulation, question answering or chart drawing. You can only take one action `python_repl_ast`, but you can write python code to call a suite of provided APIs in the simulated python shell. Print out the final answer for the the question answering task in the end.

API tools are given below:

```

get_mean: get_mean(df: 'pd.DataFrame', column: 'str') -> 'float'
- Return the mean of a column.
get_max: get_max(df: 'pd.DataFrame', column: 'str') -> 'float'
- Return the max of the column.
get_min: get_min(df: 'pd.DataFrame', column: 'str') -> 'float'
- Return the min of the column.
get_sum: get_sum(df: 'pd.DataFrame', column: 'str') -> 'float'
- Return the sum of a column.
get_count: get_count(df: 'pd.DataFrame') -> 'int'
- Return the count of rows.
get_abs: get_abs(df: 'pd.DataFrame', column: 'str') -> 'pd.DataFrame'
- Turn the values of a column into absolute values.
draw_bar: draw_bar(title: 'str', height_list: 'list[Union[int, float]]', x_labels: 'list[str]') -> 'plt' - Draw a bar chart.
draw_line: draw_line(title: 'str', x_list: 'list[Union[int, float]]', y_list: 'list[Union[int, float]]', x_labels: 'list[str]') -> 'plt'
- Draw a line chart.
draw_scatter: draw_scatter(title: 'str', x_list: 'list[Union[int, float]]', y_list: 'list[Union[int, float]]') -> 'plt'
- Draw a scatter chart.
draw_pie: draw_pie(title: 'str', number_list: 'list[Union[int, float]]', label_list: 'list[str]') -> 'plt' - Draw a pie chart.
drop_row: drop_row(df: 'pd.DataFrame', row_index: 'int') -> 'pd.DataFrame' - Drop a row indexed by 'row_index'.
drop_column: drop_column(df: 'pd.DataFrame', column: 'str') -> 'pd.DataFrame' - Drop a column named by 'column'.
insert_column: insert_column(df: 'pd.DataFrame', new_column: 'str', new_column_list: 'list') -> 'pd.DataFrame'
- Insert a new column named by 'new_column' to the table 'df'.
insert_row: insert_row(df: 'pd.DataFrame', new_row_list: 'list[Union[int, float, str]]') -> 'pd.DataFrame'
- Insert a new row to table.
calculate_percentage: calculate_percentage(numerator: 'Union[int, float]', denominator: 'Union[int, float]') -> 'str'
- Return the percentage of two numbers as a string.
edit_cell: edit_cell(df: 'pd.DataFrame', row_index: 'int', column: 'str', new_value: 'Union[int, float]') -> 'pd.DataFrame'
- Edit a cell value.
print_df: print_df(df, limited=False)
- Print out the table in a markdown format.
Set 'limited' true to only print out the first 6 rows.
filter_df: filter_df(df: 'pd.DataFrame', sql: 'str') -> 'pd.DataFrame'
- Filter the table using SQL queries.
'sql' is the SQL statement to perform the filtering operation.
'df' can be directly used as the table name in the SQL statement.
sort_column: sort_column(df: 'pd.DataFrame', column: 'str', ascending: 'bool') -> 'pd.DataFrame' - Sort the table by columns.
aggregate_df: aggregate_df(df: 'pd.DataFrame', sql: 'str') -> 'pd.DataFrame'
- Apply aggregation functions to the table using SQL queries.
'sql' is the SQL statement to perform the filtering operation.
'df' can be directly used as the table name in the SQL statement.

```

`python_repl_ast`: A Python shell. Use this to execute python commands. Input should be a valid python command. When using this tool, sometimes output is abbreviated - make sure it does not look abbreviated before using it in your answer.

Use the following format:

Question: the input question you must answer

Thought: you should always think about what to do

Action: can ONLY be `python_repl_ast`

Action Input: the action to take, should be one of [`get_mean`, `get_max`, `get_min`, `get_sum`, `get_count`, `get_abs`, `draw_bar`, `draw_line`, `draw_scatter`, `draw_pie`, `drop_row`, `drop_column`, `insert_column`, `insert_row`, `calculate_percentage`, `filter_df`, `sort_column`, `aggregate_df`, `edit_cell`, `print_df`, `convert_to_numeric`].

Observation: the result of the action
 ... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I have completed the task.
Final Answer: final answer / manipulated df / image path.

Demonstration Examples:

Task: You are given a table on world's top 10 highest mountains. Filter the table to show only the mountains with an elevation above 8,500 meters.

Table: this is the result of `print(df.to_markdown())`

	Rank	Mountain	Elevation (m)	Location
0	1	Mount Everest	8,848.86	Nepal/China
1	2	K2	8,611	Pakistan/China
2	3	Kangchenjunga	8,586	Nepal/India
3	4	Lhotse	8,516	Nepal/China
4	5	Makalu	8,485	Nepal/China
5	6	Cho Oyu	8,188	Nepal/China
6	7	Dhaulagiri I	8,167	Nepal
7	8	Manaslu	8,156	Nepal
8	9	Nanga Parbat	8,126	Pakistan
9	10	Annapurna I	8,091	Nepal

Question: Filter the table to show only the mountains with an elevation above 8,500 meters.

Thought: I need to filter the table to show only the mountains with an elevation above 8,500 meters. Firstly I need to convert the elevation column to numeric.

Action: `python_repl_ast`

Action Input:

```
convert_to_numeric(df, 'Elevation (m)')
```

Observation:

Thought: Now I can filter the table.

Action: `python_repl_ast`

Action Input:

```
filter_df(df, 'SELECT * FROM df WHERE "Elevation (m)" > 8500')
```

Observation:

Thought: I have completed the task.

Final Answer: The manipulated df is shown above.

Task: You are given a table on Oklahoma State Regents for Higher Education. Sort the table by the "Term Expires" column in ascending order.

Table: this is the result of `print(df.to_markdown())`

(Table contents are omitted due to the prompt length limit.)

Thought: I need to sort the table by the "Term Expires" column in ascending order.

Action: `python_repl_ast`

Action Input:

```
sort_column(df, 'Term Expires', True)
```

Observation:

Thought: I have completed the task.

Final Answer: The sorted table is shown above.

Query: You are given a table on List of correctional facilities in New Zealand. Filter the table to show only facilities located in Auckland.

(The table will be provided to the LLMs as a `Pandas.DataFrame` object `df`.)

Trace:

Question: Filter the table to show only facilities located in Auckland.

Thought: I need to filter the table to show only facilities located in Auckland. I can do this using the `filter_df` API.

Action: `python_repl_ast`

Action Input:

```
filter_df(df, 'SELECT * FROM df WHERE Location = "Auckland"')
```

Observation:

Empty DataFrame

Columns: [Name, Location, Opened, Gender, Security level(s), Capacity]

Index: []

Thought: The filter did not work because the location column contains more than just "Auckland". I need to modify the SQL query to use the LIKE operator to match any location containing "Auckland".

Action: `python_repl_ast`

Action Input:

```
filter_df(df, 'SELECT * FROM df WHERE Location LIKE "%Auckland%")
```

Observation:

```
Name Location Opened Gender Security level(s) Capacity
Auckland Prison                               Paremoremo, Auckland ...
Mount Eden Corrections Facility               Mount Eden, Auckland ...
Auckland Region Women's Corrections Facility  Wiri, Auckland ...
```

(Some table cells are omitted.)

Thought: I have completed the task.

Final Answer:

Name	Location
Auckland Prison	Paremoremo, Auckland
Mount Eden Corrections Facility	Mount Eden, Auckland
Auckland Region Women's Corrections Facility	Wiri, Auckland

(Some table columns are omitted.)

A.16 Translation

Instruction:

Your task is to complete a given piece of text by using a Machine Translation API. You can do so by writing `[MT(text)]` where text represents the text to be translated into English.

Demonstration Examples:

Input: He has published one book: The Supressed Man. 他出版的书是什么?

Output: `[MT(他出版的书是什么?)]`

Input: In Morris de Jonge's Jeschuah, der klassische jüdische Mann, there is a description of a Jewish writer. ¿De qué país es este autor?

Output: `[MT(¿De qué país es este autor?)]`

Query:

Input: It is within the Russian Southern Federal District. 它在哪里?

Output:

Action: `[MT(它在哪里?)]`

Continue to demonstrate:

Example 1

Context: 1982-1990 Research Associate, Faculty of Science and Technology, Keio University (Professor Kunitsugu Aramaki)

Question: ¿Cuál es el nombre del profesor?

Answer the question in English:

Kunitsugu Aramaki

Example 2

Context: The society was founded on 6 (18) August 1845 in Saint Petersburg, Russia. Prior to the Russian Revolution of 1917, it was known as the Imperial Russian Geographical Society.

Question: Welche Gesellschaft wurde 1845 gegründet?

Answer the question in English:

Imperial Russian Geographical Society

Example 3

Context: It is within the Russian Southern Federal District.

Question: `[MT(它在哪里?)]`

Answer the question in English:

MLQA Response: Russian Southern Federal District.

A.17 Wikipedia

Instruction:

Solve a question answering task with interleaving Thought, Action and Observation steps. Thought can reason about the current situation, and Action can be four types:

- (1) `Search[entity]`: The input is an entity name. The action will search this entity name on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to search next.
- (2) `Lookup[keyword]`: The input is some keywords. This action will look up in the current passage and return the next several sentences containing the keyword in current passage.
- (3) `Disambiguate[entity]`: The input is an entity name. This action will disambiguate this entity name to find other entities with similar names in Wikipedia. It will return the first five entities if they exist. If not, it will return an empty list.
- (4) `Finish[answer]`: Given all the information and thoughts, return the answer and finish the task. The answer should be as simple as possible, like one or two words instead of a sentence.

Here are some tips for you to use the tools to finish the task better.

- (1) Don't search the same entity two times since the results are always the same.
- (2) When the search action doesn't find the corresponding page, you should try to search for a similar entity.
- (3) When the search action returns a page which is not related to the question, you should disambiguate the entity to find other entities that share similar names with the current one.
- (4) You can only search one entity name in each action.

Demonstration Examples:

Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?

Thought 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.

Action 1:

`Search[Colorado orogeny]`

Observation 1: The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas. This took place from 1780 to 1650 million years ago (Mya), during the Paleoproterozoic (Statherian Period). It is recorded in the Colorado orogen, a >500-km-wide belt of oceanic arc rock that extends southward into New Mexico. The Colorado orogeny was likely part of the larger Yavapai orogeny.

Thought 2: It does not mention the eastern sector. So I need to look up eastern sector.

Action 2:

`Lookup[eastern sector]`

Observation 2: The Wyoming sector of the Colorado orogeny was formerly called the Medicine Bow orogeny. The eastern sector extends into the High Plains and is called the Central Plains orogeny. The boundary between the Colorado orogeny and the Wyoming craton is the Cheyenne belt, a 5-km-wide mylonitic shear zone that verges northward.

Thought 3: The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.

Action 3:

`Search[High Plains]`

Observation 3: High Plains refers to one of two distinct land regions.

Thought 4: This page is not related to the question. I need to disambiguate High Plains to find entities sharing similar name.

Action 4:

`Disambiguate[High Plains]`

Observation 4: Related entities to High Plains: ['High Plains (United States)', 'High Plains (Australia)', 'Altiplano (disambiguation)']

Thought 5: High Plains (United States) is most likely related to the question. I need to instead search High Plains (United States).

Action 5:

`Search[High Plains (United States)]`

Observation 5: The High Plains are a subregion of the Great Plains, mainly in the Western United States, but also partly in the Midwest states of Nebraska, Kansas, and South Dakota, generally encompassing the western part of the Great Plains before the region reaches the Rocky Mountains. The High Plains are located in eastern Montana, southeastern Wyoming, southwestern South Dakota, western Nebraska, eastern Colorado, western Kansas, eastern New Mexico. The southern region of the Western High Plains ecology region contains the geological formation known as Llano Estacado which can be seen from a short distance or on satellite maps. From east to west, the High Plains rise in elevation from around 1,800 to 7,000 ft (550 to 2,130 m).

Thought 6: High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.

Action 6:

`Finish[1,800 to 7,000 ft]`

Query: What country is the Selun located in?

Trace:

Thought 1: I need to search Selun and find the country it is located in.

Action 1:


```
Search[Selun]
```

Observation 1: The Selun is one of the peaks of the Churfirsten range, located in the Appenzell Alps. It lies between the valley of Toggenburg and Lake Walenstadt in the canton of St. Gallen. The summit is easily accessible by a trail on the northern side.. The peak is named for the extended alpine pasture Selunalp to the peak's north-west, situated above c.

Thought 2: The paragraph does not mention the country. I need to look up "country".

Action 2:

```
Lookup[country]
```

Observation 2: No more results.

Thought 3: The paragraph does not mention the country, so I need to search Churfirsten range instead and find the country it is located in.

Action 3:

```
Search[Churfirsten range]
```

Observation 3: Could not find Churfirsten range. Similar: ['Churfirsten', 'Walensee', 'Chäserrugg', 'Point Break (2015 film)', 'North-Eastern Swiss Alps (category Mountain ranges of Switzerland)'].

Thought 4: Churfirsten is the mountain range the Selun is located in. I need to search Churfirsten and find the country it is located in.

Action 4:

```
Search[Churfirsten]
```

Observation 4: Churfürsten is a mountain range in the Canton of St. Gallen, Switzerland. They form the natural boundary between the canton's Toggenburg and Sarganserland districts.. They are the southernmost range of the Appenzell Alps, separated from the Glarus Alps by the Seez river and Walensee.. They consist of a limestone ridge running east to west, with the individual peaks formed by erosion..

Thought 5: The Churfirsten range is located in the Canton of St. Gallen, Switzerland. So the answer is Switzerland.

Action 5:

```
Finish[Switzerland]
```

A.18 Database

Instruction:

System: You are a Data Agent that helps users to efficiently query and manage the data stored in database systems. You can call the following commands to fulfill jobs like schema generation, sql translation, query rewriting, data lookup.

Goals

Please accurately generate the sql query based on the schema and user description, efficiently execute the sql query on given database, and print out the corresponding query results in suitable format.

Commands

(1) `get_database_schema(db_name:str)` obtains the schema of the given database.

(2) `translate_text_to_sql(description:str)` translates the input text string into a sql based on the schema.

(3) `rewrite_sql(sql:str)` rewrites the input sql into an execution-efficient sql query.

(4) `select_database_data(query:str)` reads the data stored in the database based on the sql query.

Here are some tips for you to use the commands to finish this task better.

(1) Internet access for searches and information gathering.

(2) Long term memory management.

(3) If a command is successfully called for two times, do not call the same command again;

(4) Do not use any image in the output.

Demonstration Example:

Task: Fetch the database results based on the following descriptions: "Retrieve the comments of suppliers. The results should be sorted in descending order based on the comments of the suppliers".

Plan:

- Use the `get_database_schema` command to retrieve the schema
- Use the `translate_text_to_sql` command to write the SQL query based on the given description
- Use the `select_database_data` command to retrieve the data based on the SQL query
- Count the number of rows in the query results and output the query results.

Thought 1: I need to get the database schema for the database named `tpch` and generate an SQL query based on the given description. I should start by using the `get_database_schema` command to retrieve the schema.

Reasoning 1: I need to retrieve the schema to know the names of the tables and columns in the database. This will enable me to write the SQL query accurately.

Action 1:

```
get_database_schema('tpch')
```

Observation 1: Command `get_database_schema` returned: “The database schema is: { supplier : [suppkey, name, address, nationkey, phone, acctbal, comment], ... }”

Thought 2: Now that I have the database schema, I will use the `translate_text_to_sql` command to generate the SQL query based on the given description and schema, and take the SQL query as the input of the `rewrite_sql` and `select_database_data` commands.

Reasoning 2: I need to generate the SQL query accurately based on the given description. I will use the `translate_text_to_sql` command to obtain the SQL query based on the given description and schema.

Action 2:

```
translate_nlp_to_sql('Retrieve the comments of suppliers. The results
should be sorted in descending order based on the comments of the
suppliers.')
```

Observation 2: Command `translate_text_to_sql` returned: “SELECT comment FROM supplier ORDER BY comment DESC”.

Thought 3: Now that I have the database schema and the sql query does not need to further rewrite, I will use the `select_database_data` command to retrieve the data from the database.

Reasoning 2: I will use the `select_database_data` command to retrieve the data from the database because it is the next logical step.

Action 3:

```
select_database_data('SELECT comment FROM supplier ORDER BY comment DESC')
```

Final Answer: Command `select_database_data` returned: "The number of result rows is: 100000"

```
| ..... comment ..... |
| ... regular packages across the enticing accounts ... |
| ... quietly among the regular ideas. fluffily fi ... |
| ..... |
```

Task:

Retrieve the comments in the supplier table where the partkey column in the part table matches the partkey column in the partsupp table, the availqty column in the partsupp table equals 6331, the type column in the part table is greater than 'LARGE POLISHED NICKEL', and the retailprice column in the part table is less than 1758.76.

Trace:

Task: Fetch the database results based on the `task` description.

Plan:

- Use the `get_database_schema` command to retrieve the schema
- Use the `translate_text_to_sql` command to write the SQL query based on the given description
- Use the `rewrite_sql` command to rewrite the SQL query
- Use the `select_database_data` command to retrieve the data based on the rewritten SQL query
- Count the number of rows in the query results and output the query results.

Thought 1: I need to get the database schema for the `tpch` database and generate an SQL query based on the given description. I should start by using the `get_database_schema` command to retrieve the schema.

Reasoning 1: I need to retrieve the schema to know the names of the tables and columns in the database. This will enable me to write the SQL query accurately.

Action 1:

```
get_database_schema('tpch')
```

Observation 1: Command `get_database_schema` returned: “The database schema is: { supplier: [suppkey, name, address, nationkey, phone, acctbal, comment], partsupp: [partkey, suppkey, availqty, supplycost, comment], ...}”

Thought 2: Now that I have the database schema, I will use the `translate_text_to_sql` command to generate the SQL query based on the given description and schema, and take the SQL query as the input of the `rewrite_sql` and `select_database_data` commands.

Reasoning 2: I need to generate the SQL query accurately based on the given description. I will use the `translate_text_to_sql` command to obtain the SQL query based on the given description and schema.

Action 2:

```
translate_nlp_to_sql('...') # input: the task description
```

Observation 2: Command `translate_text_to_sql` returned: “SELECT comment FROM supplier JOIN partsupp ON supplier.suppkey = partsupp.suppkey JOIN part ON partsupp.partkey = part.partkey WHERE partsupp.availqty = 6331 AND part.type > 'LARGE POLISHED NICKEL' AND part.retailprice < 1758.76”.

Thought 3: Now that I have the SQL query, I will then use the `rewrite_sql` command to rewrite the SQL query to obtain the database results of the rewritten SQL query.

Reasoning 2: I need to rewrite the SQL query before retrieving the data from the database based on the SQL query.

Action 3:

```
rewrite_sql('...') # input: the sql query
```

Observation 3: Command `rewrite_sql` returned: “SELECT comment FROM supplier s JOIN (SELECT * FROM partsupp WHERE partsupp.availqty = 6331) ps ON s.supkey = ps.supkey JOIN (SELECT * FROM part WHERE part.type > 'LARGE POLISHED NICKEL' AND part.retailprice < 1758.76) p ON ps.partkey = p.partkey”.

Thought 4: Now that I have the rewritten sql query, I will use the `select_database_data` command to retrieve the data from the database.

Reasoning 4: I will use the `select_database_data` command to retrieve the data from the database based on the rewritten SQL query.

Action 4:

```
select\_database\_data(`...`) # input: the rewritten query
```

Final Answer: Command `select_database_data` returned: "The number of result rows is: 394"

```
| ..... comment ..... |
| ... quickly unusual orbits haggle carefully accounts .... |
| ... deposits wake above the even, pending pinto beans ... |
| ..... |
```