TPTU-v2: Boosting Task Planning and Tool Usage of Large Language Model-based Agents in Real-world Systems

Yilun Kong[†‡]
kongyilun@sensetime.com

Jingqing Ruan[†‡]
ruanjingqing@sensetime.com

Yihong Chen[†‡]
chenyihong@sensetime.com

Bin Zhang[†‡]
zhangbin11@sensetime.com

Tianpeng Bao[†]
baotianpeng@sensetime.com

Shiwei Shi[†]
shishiwei@sensetime.com

Guoqing Du[†]
duguoqing@sensetime.com

Xiaoru Hu[†]
huxiaoru@sensetime.com

Hangyu Mao[†✉]
maohangyu@sensetime.com

Ziyue Li
zlibn@connect.ust.hk

Xingyu Zeng
zengxingyu@sensetime.com

Rui Zhao
zhaorui@sensetime.com

SenseTime Research

Abstract

Large Language Models (LLMs) have demonstrated proficiency in addressing tasks that necessitate a combination of task planning and the usage of external tools that require a blend of task planning and the utilization of external tools, such as APIs. However, real-world complex systems present three prevalent challenges concerning task planning and tool usage: (1) The real system usually has a vast array of APIs, so it is impossible to feed the descriptions of all APIs to the prompt of LLMs as the token length is limited; (2) the real system is designed for handling complex tasks, and the base LLMs can hardly plan a correct sub-task order and API-calling order for such tasks; (3) Similar semantics and functionalities among APIs in real systems create challenges for both LLMs and even humans in distinguishing between them. In response, this paper introduces a comprehensive framework aimed at enhancing the Task Planning and Tool Usage (TPTU) abilities of LLM-based agents operating within real-world systems. Our framework comprises three key components designed to address these challenges: (1) the *API Retriever* selects the most pertinent APIs for the user's task among the extensive array available; (2) *LLM Finetuner* tunes a base LLM so that the finetuned LLM can be more capable for task planning and API calling; (3) the *Demo Selector* adaptively retrieves different demonstrations related to hard-to-distinguish APIs, which is further used for in-context learning to boost the final performance. We validate our methods using a real-world commercial system as well as an open-sourced academic dataset,

[†]These authors contribute equally to this work.
[‡]These authors work as research interns at SenseTime Research.
[✉]The corresponding author.

and the outcomes clearly showcase the efficacy of each individual component as well as the integrated framework.

1 Introduction

Large language models (LLMs) have exhibited remarkable prowess in natural language processing (NLP) [1–3], encompassing language understanding [4, 5], reasoning [6, 7], and program synthesis [8, 9].

However, leveraging LLMs for complex tasks presents formidable challenges. On one hand, LLMs inherently possess limitations in their capabilities. They have been shown to struggle with solving logical problems such as mathematics, and their training data can quickly become outdated as the world evolves. Instructing LLMs to utilize external tools such as calculators, calendars, or search engines can help prevent them from generating inaccurate information and aid them in effectively addressing problems. On the other hand, integrating these models into complex systems transcends mere task understanding. It demands the ability to break down intricate tasks, manipulate various tools, and engage with users in effective interactions. Several research endeavors, known as LLM-based AI Agents [10, 11], such as AutoGPT [1], BabyAGI [2], and GhatGPT-plugins [3], have made advancements by employing LLMs as central controllers. These endeavors automatically decompose user queries into sub-tasks, execute low-level tool (API) calls for these sub-tasks, and ultimately resolve the overarching problem.

Despite these advances, LLM-based agents still grapple with pressing challenges in real-world applications. Firstly, real-world systems usually have a vast number of APIs, making it impractical to input descriptions of all APIs into the prompt of LLMs due to the token length limitations. Secondly, the real system is designed for handling complex tasks, and the base LLMs often struggle to correctly plan sub-task orders and API-calling sequences for such tasks. Thirdly, the real system is primarily designed around a core purpose, and as a result, certain APIs may overlap and exhibit similar semantics and functionality, creating difficulty in differentiation for both LLMs and humans. How to address these issues could be the critical step for LLM-based Agents towards omniscience and omnipotence in the real world.

In this paper, we propose a framework to improve the **T**ask **P**lanning and **T**ool **U**sing (TPTU) [12, 13] abilities of LLM-based agents in the real-world systems. Compare to our TPTU-v1 [12, 13], our new framework consists of three key components to address the above three challenges: (1) **API Retriever** recalls the APIs that are most relevant to the user's task from all APIs. The descriptions of these filtered APIs can then be input into LLM as prompts, allowing the LLM to understand and make accurate choices within the filtered API set. (2) **LLM Finetuner** tunes a base LLM so that the finetuned LLM can be more capable of task planning and API calls, especially for domain-specific tasks. (3) **Demo Selector** adaptively retrieves different demonstrations related to hard-to-distinguish APIs, which is further used for in-context learning so that LLM can distinguish the subtle differences in the functions and usages of different APIs. Our main contributions can be summarized as follows:

1. We identify three practical challenges that LLM-based agents face when it comes to task planning and tool usage in real-world scenarios.
2. In response to the three challenges mentioned above, we propose an advanced framework composed of three key components: API Retriever, LLM Finetuner, and Demo Selector.
3. Extensive experiments in real-world commercial systems demonstrate the effectiveness of each component and the integrated framework, where the tasks are highly complex and closely intertwined with people's lives. We also validate our methods with open-sourced academic datasets.

[1] `https://github.com/Significant-Gravitas/Auto-GPT`
[2] `https://github.com/yoheinakajima/babyagi`
[3] `https://openai.com/blog/chatgpt-plugins`

2 Methodology

In response to the typical challenges of deploying LLMs within intricate real-world systems, we propose a comprehensive framework that fundamentally bolsters the capabilities of LLMs in Task Planning and Tool Usage (TPTU). This section first introduces our proposed framework, which systemically integrates three specialized components: an API Retriever, an LLM Finetuner, and a Demo Selector. Subsequently, we delve into a comprehensive description of each component, elucidating their unique contributions to the overall framework.

2.1 Framework Overview

Our comprehensive framework is engineered to enhance the capabilities of LLMs in Task Planning and Tool Usage (TPTU) within complex real-world systems. The framework is meticulously designed to address three core challenges: the extensive number of APIs in real-world systems, the complexity of correct task and API call sequencing, and the difficulty in distinguishing between APIs with overlapping functionalities.

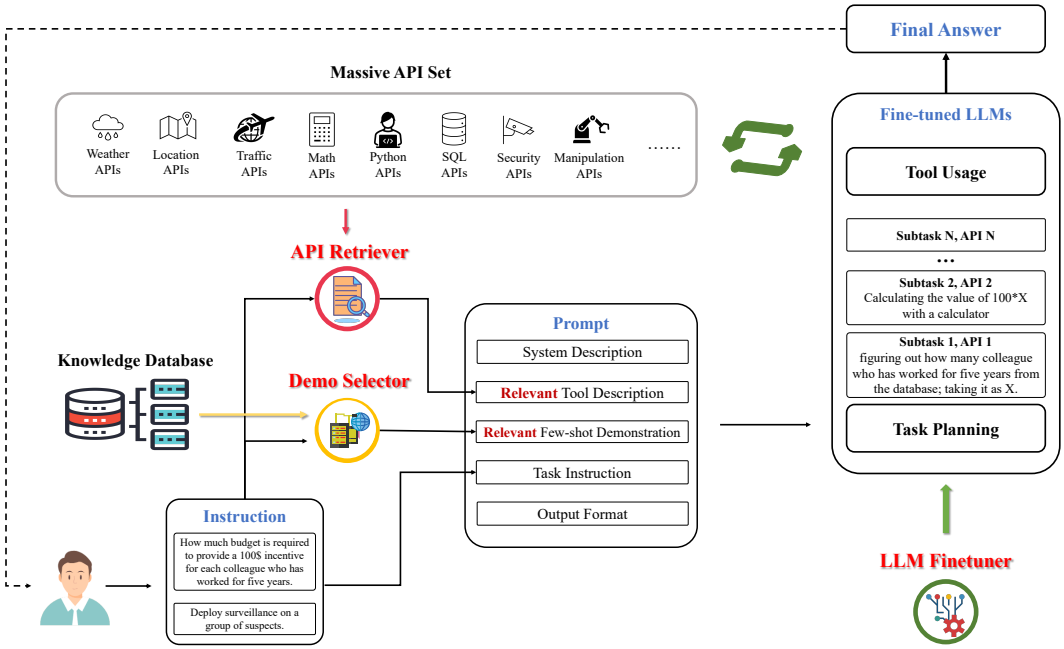


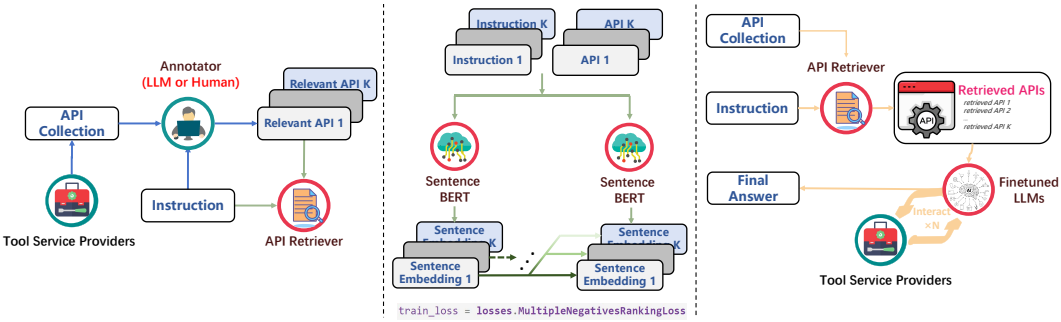
Figure 1: The proposed framework.

The framework is composed of three pivotal components, depicted in Figure 1.

1. **API Retriever**: This component navigates through an extensive array of APIs to retrieve the most relevant ones based on the user's task. It employs an advanced embedding search technique to understand the semantics of the task and match it with the correct APIs, leveraging a rich Knowledge Database and an API Collection to ensure relevance and accuracy.
2. **LLM Finetuner**: This subsystem fine-tunes a base LLM with a meticulously curated dataset, enhancing the model's ability to plan tasks and execute API calls efficiently. The fine-tuning process is informed by diverse datasets, including ones specifically created to increase prompt diversity and address both single-step and multi-step API interactions.
3. **Demo Selector**: The Demo Selector dynamically retrieves demonstrations related to hard-to-distinguish APIs, facilitating in-context learning for the LLM. This allows the model to discern subtle functional differences between APIs, crucial for generating precise outputs, especially when dealing with similar APIs.

2.2 API Retriever

In real-world systems, there exists a massive number of APIs for problem-solving, which poses a severe challenge for the integration of LLMs. On the one hand, the token limitations inherent to LLMs impede the inclusion of all API descriptions in the model's prompt, potentially surpassing the maximum token length. On the other hand, even when the inclusion of numerous APIs does not breach these token constraints, the presence of excessive, task-irrelevant API information can interfere with the model's capacity for accurate planning and answer generation, thereby hindering its operational efficiency. To surmount these challenges, we have developed a novel model explicitly trained to select the APIs of utmost relevance to the task at hand, shown in Figure 2. Building on the overview of the API Retriever framework, we will now give a detailed description of the data collection, training, and inference process.



(a) Data Construction (b) Trainning (Finetuning Sentence BERT) (c) Inference

Figure 2: The proposed framework of API Retriever.

2.2.1 Data Collection

The foundation of the API Retriever's effectiveness lies in a rigorous data collection process. First, we have collected a comprehensive set of APIs provided by a multitude of external tool services. This collection forms the substrate upon which our model is trained. To ensure that our system understands the relevance of different APIs to various user queries (instructions), we have instituted a particular annotation process. In this process, human experts, or LLMs, analyze complex user instructions (or tasks) and identify the APIs that are necessary for resolving these instructions. This hybrid approach not only enriches our dataset with human expertise but also benefits from the scale and efficiency of LLMs in processing large quantities of data. By combining the precision of human annotations with the breadth of LLMs' processing abilities, we create a dataset that is both qualitatively rich and quantitatively vast, laying a solid foundation for the subsequent training phase of the API Retriever. We give a detailed demonstration of the dataset in Figure 3.

2.2.2 Training

Following the collection of this annotated data, the training of the API Retriever is conducted to maximize the relevance of the retrieved APIs to the task instruction of users. The training framework for the API Retriever is depicted as a dual-stream architecture employing Sentence-BERT [14], a variant of the BERT [4] model optimized for generating sentence embeddings. The training process utilizes pairs of instructions and their corresponding APIs, which are denoted as *Instruction 1* through *Instruction K* and *API 1* through *API K*, respectively.

Each instruction and API description is processed through its own Sentence-BERT model to obtain semantically rich embeddings. This means that for each instruction-API pair, we generate two separate embeddings that encapsulate the semantic essence of the text. The embeddings for the instructions are labeled as *Sentence Embedding 1* to *Sentence Embedding K*, and similarly, the embeddings for the APIs follow the same notation.

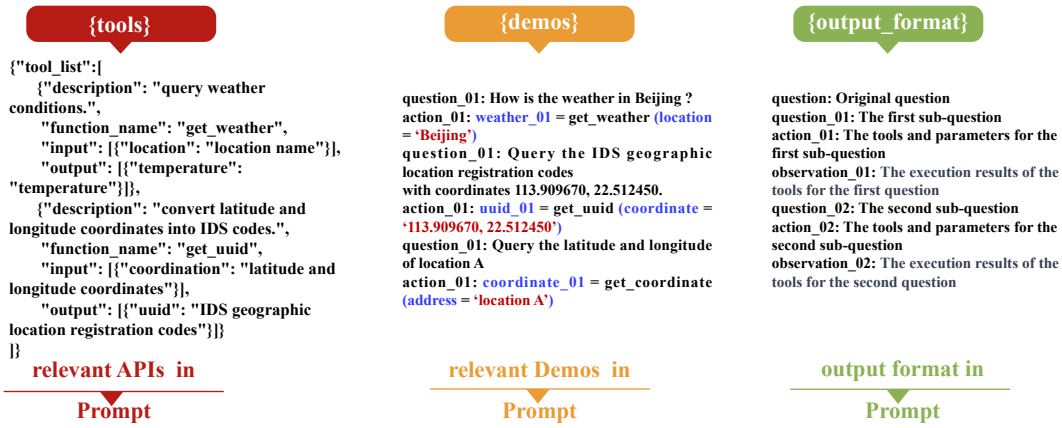


Figure 3: The detailed demonstration of the dataset for the API Retriever.

The framework employs a training objective known as the Multiple Negatives Ranking Loss[4] [15]. This loss function is designed to contrast a positive pair (a correct association between instruction and an API) against multiple negative pairs (incorrect associations). The goal is to minimize the distance between the embeddings of correct instruction-API pairs while maximizing the distance between the embeddings of incorrect pairs. This goal can formulated as follows.

$$\mathcal{L} = -\frac{1}{K}\sum_{i=1}^{K}\log\frac{e^{sim(s_i,s_i^+)}}{e^{sim(s_i,s_i^+)}+\sum_{j\neq i}e^{sim(s_i,s_j^-)}}, \tag{1}$$

where s_i and s_i^+ denote the *Sentence Embedding i* and the corresponding *Sentence Embedding i* of the API, respectively. $sim(\cdot)$ is the similarity function that calculates the similarity between two vectors (embeddings in this context). Our choice for $sim(\cdot)$ is the cosine similarity, which measures the cosine of the angle between two vectors u and v, defined as follows.

$$sim(u,v) = \frac{u \cdot v}{||u||||v||}, \tag{2}$$

where $u \cdot v$ is the dot product of vectors, and $||\cdot||$ denotes Euclidean norms (or magnitudes) of the vectors.

During training, this encourages the model to learn a representation space where instructions and their relevant APIs are closer to each other, thus facilitating more accurate retrieval of APIs in response to new instructions.

In summary, the Sentence-BERT models in this framework are fine-tuned to learn the semantic relationships between user instructions and APIs, enabling the API Retriever to discern and prioritize the most relevant APIs for a given task based on their learned embeddings.

2.2.3 Inference

The inference diagram illustrates the process that integrates the API Retriever and LLMs with the objective of generating a final answer to a given instruction.

The process commences with an *Instruction*: a user's query or task that needs to be addressed. This *Instruction* is fed into the API Retriever, a component that has been meticulously trained to recognize and select the most relevant APIs from an extensive API Collection. The API Retriever evaluates the instruction, determines the relevant APIs needed to fulfill the task, and retrieves a subset of APIs, denoted as *retrieved API 1* to *retrieved API K*.

Once the relevant APIs are retrieved, they are fed into the tool-level prompt for LLMs to select the accurate APIs to solve certain instructions. It is important to note that there might be multiple

[4] https://www.sbert.net/docs/package_reference/losses.html#multiplenegativesrankingloss

interactions ("Interact $\times$ N") between the LLMs and the Tool Service Providers, which are the actual endpoints of the APIs, indicating that the LLMs may call multiple APIs multiple times to gather the information needed.

Finally, after the LLMs have interacted with the tool service providers as required, they summarize the information gathered from the APIs to construct a "Final Answer". This answer is expected to be a comprehensive response to the original instruction, showcasing the system's ability to understand, retrieve, and apply relevant information to solve complex, real-world problems.

2.3 LLM Finetuner

While open-sourced LLMs possess strong capabilities, they often encounter limitations due to a lack of specificity and adaptability within complex, specialized, real-world domains. Furthermore, certain models may fall short in their generative abilities, struggling to yield high-quality outputs when tasked with challenges. To address these issues, we shift our approach from pioneering new fine-tuning methods to *concentrating on the development of a dataset, expressly curated to enhance the fine-tuning process for real-world systems*. In this context, we will also share some insights during the fine-tuning procedure, providing a clearer understanding of its influence on model performance.

Building upon the foundation established by the introduction, we delve into the fine-tuning of our LLMs using the prevalent method known as Supervised Fine-Tuning (SFT). This mainstream approach to fine-tuning involves adjusting the pre-trained weights of an LLM on a dataset that is labeled with the correct outputs for given inputs. SFT is particularly effective for enhancing model performance in specific domains or tasks, as it steers the model toward the desired output using the provided supervisory signals.

For our fine-tuning process, we have constructed and analyzed three distinct datasets, each representing a unique fine-tuning paradigm:

1. **Training Set v1**: Born out of a need for datasets that accurately mirror real-world scenarios, this initial dataset was constructed by carefully selecting genuine cases, eliminating ineffective data and duplicate cases. Its motivation lies in grounding the SFT in reality, aligning the LLM's understanding with the true data distribution found in practical real-world use. The dataset serves as a preliminary step towards tuning the LLM to adapt to real-world data distribution.

2. **Training Set v2**: This dataset is selectively compiled based on prompt functionality, encompassing a total of 745 entries. It is augmented with system-level prompts that include a comprehensive list of features and their descriptions. These enriched prompts serve to provide the LLM with a more detailed understanding of each API's capabilities and constraints. By incorporating a detailed functionality list and descriptions within the prompts, we aim to enhance the model's ability to generate responses that not only match the input query semantically but also align closely with the functional scope of the available APIs. This structured approach to prompt design is crucial for enabling the LLM to navigate the API space with greater precision, particularly when dealing with complex, multi-faceted user requests.

3. **Training Set v3**: Recognizing the limitations of our previous dataset, which predominantly featured single-step API calls and suffered from a lack of prompt diversity, we sought to more closely cover real-world scenarios. Training Set v3 was thus meticulously engineered to bridge this domain gap, comprising 660 question-and-answer pairs that reflect the complexity of actual use cases. (1) For prompt diversity, we employ various data augmentation on prompts, e.g., randomly shuffling API orders and adding irrelevant APIs, thus decreasing the risk of over-fitting and enhancing the robustness of the LLM. (2) For instruction diversity, we replace the original user instruction with similar-meaning instructions by means like rewriting-by-LLMs, synonym substitution, and loop-back translation. This makes LLMs more robust to different user instructions during inference. (3) For output diversity, set v3 intentionally includes a balanced mix of 390 single-step API interactions, which solidify the foundational understanding of API functionalities, and an additional 270 multi-step API calls, which introduce the LLM to more complex sequences of operations that are commonly encountered in practice.

Each dataset is intended to incrementally refine the LLM's ability to parse user inputs, understand the context, and generate precise API calls. Finetuning LLMs on these datasets can enhance the ability of LLMs to solve specific real-world tasks. The analysis of model performance across these datasets provides valuable insights into the effects of prompt diversity and task complexity on the LLM's fine-tuning efficiency and its eventual real-world applicability. By systematically evaluating the model's output against these varied fine-tuning paradigms, we enhance its competency in delivering high-quality, contextually appropriate responses in the domain of API interaction.

The insights obtained from the iterative development of these datasets demonstrate the critical importance of dataset quality and construction in the fine-tuning process. With each successive version, we observed measurable improvements in the LLM's performance, underscoring the direct impact that well-constructed training data has on the model's ability to handle real-world tasks. It is not merely the quantity of data but the relevance, cleanliness, and alignment with actual usage patterns that drive the efficacy of fine-tuning, leading to models that are not only more versatile but also more reliable when deployed in complex real-world applications.

2.4 Demo Selector

The Demo Selector framework, as shown in Figure 4, plays a crucial role in enhancing the ability of finetuned LLMs to differentiate between APIs with similar functionalities and semantics [5]. Usually, the quality of demonstrations has a very positive influence on promoting the ability of LLMs to disassemble complex tasks. Here is a detailed description of the main workflow and functionality of the Demo Selector, guided by the provided knowledge and the information depicted in Figure 4.

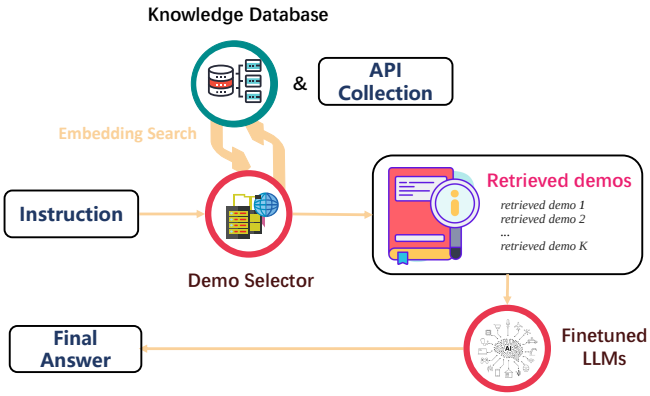


Figure 4: The proposed framework of the Demo Selector.

The Demo Selector is engineered to dynamically retrieve various demonstrations pertinent to APIs that are challenging to distinguish due to their overlapping features. The main workflow begins with an "Instruction", which represents a user's query or command that necessitates the utilization of one or more APIs.

Upon receiving an instruction, the Demo Selector interacts with two critical resources: the "Knowledge Database" and the "API Collection". The Knowledge Database contains structured information that could include API documentation, usage examples, and other relevant data that aids in understanding the context and details of each API. The API Collection, on the other hand, comprises the actual API endpoints and their associated metadata.

Then, an embedding searching process is employed to facilitate the retrieval of relevant demonstrations (demos) for a given user query.

[5]The APIs may have similar semantics and functionality because (1) the real system is primarily designed around a core purpose, so some APIs are relevant; (2) when API retriever is used, the retrieved APIs could be more semantically similar.

1. **Embedding Generation**. Initially, the user's query Q and demos from the knowledge database D are transformed into vector representations, known as embeddings. Let $emb(Q)$ denote the embedding of the user query, and $emb(D_i)$ represent the embedding of the i-th demo in the database, where i ranges from 1 to the total number of examples N. Here, we use Sentence-Bert [14] as the tool to generate embeddings.
2. **Similarity Thresholding**. We define a similarity threshold Δ to determine the relevance of each demo. The similarity measure $sim(emb(Q), emb(D_i))$ is computed between the query embedding and each example embedding. This similarity could be calculated using cosine similarity as $sim(emb(Q), emb(D_i)) = \frac{emb(Q) \cdot emb(D_i)}{\|emb(Q)\| \|emb(D_i)\|}$, where $\cdot$ denotes the dot product of the two embeddings, and $\| \cdot \|$ represents the L2 norm.
3. **Top-k Demo Retrieval**. If the similarity measure for any example exceeds the threshold $sim(emb(Q), emb(D_i)) > \Delta$, we proceed to select the top-k most similar demos $\{D_{top_1}, D_{top_2}, ..., D_{top_k}\}$ based on their similarity scores. These are regarded as subtask-level demos as they are closely related to the specific task at hand.
4. **Fallback to API-Level Demos**: In cases where no example exceeds the similarity threshold $\forall i, sim(emb(Q), emb(D_i)) \leq \Delta$, the process defaults to retrieving demos from the API collection. This involves searching for relevant API-level demos that are aligned with the broader context of the query rather than specific subtask details.

The core functionality of the Demo Selector lies in its adaptability and precision in identifying the most relevant demonstrations for a given task query, ensuring that the LLM is provided with the most contextually appropriate examples for its operation. This process seamlessly prioritizes the retrieval of subtask-level demos that are highly relevant when available, but it can also efficiently fall back on more generalized API-level demos when specific examples do not meet the similarity threshold. By sifting through embeddings and discerning the nuanced differences in API functionalities, the Demo Selector is capable of selecting from a range of demonstrations, labeled as *retrieved demo 1* to *retrieved demo K*. These context-rich examples are instrumental in illustrating how similar APIs can be distinctively applied, thereby significantly enhancing the LLM's performance in executing complex tasks.

Finally, the interaction between the Demo Selector and the finetuned LLMs leads to the generation of a final answer, which is the LLMs' response to the original instruction, informed by the nuanced understanding gained from the demonstrations.

3 Experiments

In this section, we present an experiment designed to rigorously evaluate the efficacy of our proposed framework, with a particular focus on the API Retriever, the LLM Finetuner, and the Demo Selector components. Our experimental methodology is structured to test the system's performance in a real-world context and an open-source challenge.

We begin by detailing the experimental setup, including the datasets employed. This is followed by a series of experiments that systematically assess each component's contribution to the overall functionality of the system. Through a combination of quantitative and qualitative analyses, we aim to demonstrate not only the performance improvements our system achieves over existing approaches but also the specific capabilities it brings to complex task planning and API interaction scenarios.

3.1 Datasets

Anonymous Real-world Scenario. Diverging from the current scholarly focus on studying the ability to choose the right APIs from a plethora of APIs encompassing various functionalities, in real-world systems, more common and challenging problems often revolve around a few core purposes. It entails choosing the most suitable API from a few dozen APIs, which are closely related in semantics but differ in usage, such as required parameters. Therefore, we constructed a specialized dataset that is composed of 45 APIs revolving around 11 core functionalities, based on a real commercial security system. Note that despite the total number of APIs being only 45, real-world tasks involve different planning trajectories of APIs and their parameters. For example, some trajectories can involve 9 APIs, and the average length of API trajectories is 3.5, which is longer than many open-source

datasets [16–18]. The training dataset has been described in Section 2.3. As for the testing dataset, we collected 100 questions for evaluation. Although the number of testing questions is not large, the quality is high. Our product-side colleagues assisted us in collecting this data, including simple questions with fewer than 10 words, as well as challenging questions with more than 100 words. The careful selection of testing questions ensures that they accurately reflect real-world usage scenarios.

Open-source Scenario. To ensure the generalizability of our approach across a broader spectrum of tasks and its capability to select appropriate APIs from a myriad of options, we also perform experiments on an open-source dataset, ToolBench[16], which contains 16000+ real-world APIs spanning 49 application categories. Besides the variety and quantity of APIs, it is also well conducted with both single-tool and multi-tool scenarios, as well as several multi-step reasoning traces for each query. Thus, ToolBench can simulate a real-world system, and experiments on this dataset can further demonstrate the performance of our framework in complex real-world tasks and its generalization ability across different scenarios. In order to manage the evaluation cost-effectively, we employed a random sampling approach to select 10,000 questions from ToolBench. These questions were then split into three datasets: training, validation, and testing, using a ratio of 7:1:2 respectively. This division allows us to train and fine-tune our models on a substantial amount of data while reserving a separate portion for thorough validation and reliable testing.

3.2 Experiment on Real-world Scenario

In our anonymous real-world scenario, we conduct tests to evaluate the effectiveness of the proposed modules in our framework. We begin by assessing the capability of the API retriever on our dataset, achieving a Recall@5 of 84.64% and Recall@10 of 98.47% in Table 1. These results verify the effectiveness of our method, demonstrating a high level of precision in retrieving relevant APIs, which is crucial for the subsequent task execution phase.

Table 1: The results of API Retriever on Real-world Scenario

Approaches	Recall@5	Recall@10
API Retriever	84.64%	98.47%

Table 2: Performance comparison on Real-world Scenario

Approaches	Execution Accuracy
base LLM (no demos and oracle APIs)	38.89%
base LLM (no demos and oracle APIs) + API retriever	43.33%
base LLM (no demos and oracle APIs) + Demo selector	95.55%
finetuned LLM + API retriever	80%
finetuned LLM + API retriever + Demo selector	**96.67%**

Moving to the task execution tests, the results are presented in Table 2. We choose InternLM [19], a sophisticated language model developed by Shanghai AI Lab, as our evaluated LLM. The term "base LLM" refers to the execution of prompts that do not include demonstrations and utilize the smallest set of Oracle APIs, meticulously selected by human experts. Intuitively, one might assume that manually selected Oracle APIs would outperform the results obtained using our API Retriever. However, contrary to this expectation, our method yields comparable performance. This observation can be attributed to the significant influence of the API order in the prompt on the decisions made by the Language Model (LLM). The relative positioning of APIs within the prompt can have a substantial impact on the LLM's understanding and subsequent decision-making process. The order in which APIs are presented can affect the LLM's interpretation of the context and the relationships between different APIs, ultimately influencing its output. This phenomenon has been previously corroborated

by experimental findings in the literature [20]. Furthermore, in complex scenarios, relying solely on human expertise for precise API selection can be inadequate. It might be a promising approach to automatically retrieve the appropriate API sets.

Regarding the benefits of fine-tuning, the data clearly demonstrates its advantages. The finetuned LLM combined with the API Retriever achieves an 80% execution accuracy, significantly higher than the base LLM's performance. This improvement can be attributed to the fine-tuning process, which tailors the LLM more closely to the specifics of the real-world task. It enhances the model's understanding of the context, leading to more accurate and contextually appropriate API calls.

The highest performance is observed when combining the finetuned LLM with both the API Retriever and the Demo Selector, achieving an impressive 96.67% execution accuracy. This result underscores the effect of integrating fine-tuning with our sophisticated API retrieval and demonstration selection mechanisms. The Demo Selector, in particular, seems to have a substantial impact, likely due to its ability to provide context-rich examples that guide the LLM in making more informed decisions, especially in scenarios involving similar or complex APIs.

In conclusion, our experiments in a real-world setting validate the efficacy of our proposed framework, highlighting the importance of each component and the added value of fine-tuning in enhancing LLM performance for practical applications.

3.3 Experiment on Open-source Scenario

In the open-source scenario, we tailor our evaluation to focus primarily on the impact of fine-tuning and the API Retriever, considering that building demonstrations for this context do not significantly contribute to addressing real-world problems. Therefore, the assessment of the Demo Selector is omitted in this scenario.

Initially, we have trained the API Retriever specifically for this scenario, achieving a recall rate of 76.9%. However, due to the relatively massive nature and high similarity of APIs in this open-source environment, the recall is not as high as expected, which poses a challenge for subsequent performance evaluations.

Table 3: Performance comparison on Open-source Scenario

Approaches	Execution Accuracy
base LLM	76.67%
base LLM + API retriever	53.3%
finetuned LLM + API retriever	86.7%

As shown in Table 3, the execution accuracy of the base LLM stands at 76.67%. Interestingly, the introduction of the API Retriever results in decreased performance, dropping to 53.3%. This decline is attributable to several factors. First, the low recall of the API Retriever introduces cumulative errors in the decision-making process. In environments where APIs are relatively massive and highly similar, the increasing complexity of the API Retriever may not align well with task requirements, potentially leading to less optimal API selections. Second, if the API Retriever is trained on a dataset that does not adequately represent the diversity of the open-source scenario, it leads to overfitting. As a result, the API Retriever performs well on training data but poorly generalizes to the broader range of real-world tasks in the evaluation.

Upon implementing fine-tuning in this scenario, an enhancement in performance is observed, with the finetuned LLM combined with the API Retriever reaching an execution accuracy of 86.7%. This improvement underscores the effectiveness of fine-tuning in adapting the LLM to the specific characteristics and challenges of the open-source environment. The fine-tuning process likely helps the model better understand the nuances of the available APIs and how they correlate with different tasks, resulting in more accurate API calls and decision-making.

In summary, the open-source scenario highlights the nuanced impacts of our framework's components. It reveals the importance of aligning the capabilities of tools like the API Retriever with the specific

demands of the environment and demonstrates the substantial benefits that fine-tuning brings in enhancing model performance in a less complex API ecosystem.

4 Related Work

The remarkable capacity for using tools has facilitated the transcendence of human innate physical and cognitive limitations, enhancing our ability to comprehend, plan, and address complex tasks. In turn, the human aptitude for understanding and planning tasks contributes to the judicious selection and usage of appropriate tools. Recently, the swift evolution of LLM has rendered it viable to employ specialized tools and decompose intricate tasks like humans, which inspired significant potential in addressing real-world tasks. Substantial research has been proposed to investigate task planning and tool usage based on LLM separately, however, research that combines these abilities to mutually enhance each other is relatively scarce. TPTU[12] proposes a complete framework that enhances the agent's ability in task planning and tool utilization for addressing complex tasks. AgentTuning[21] comprehensively considers various capabilities of LLM, not only task planning and tool usage, enhancing the generalized agent capabilities of open-source LLMs themselves while ensuring their general capabilities are not compromised. Some excellent reviews also systematically discuss various aspects of LLM-based AI Agents [10, 11].

4.1 Task Planning

LLMs are pre-trained on huge text corpora and present significant common sense reasoning and multi-task generalization abilities. Prompting is a highly effective method for further harnessing the intrinsic capabilities of LLMs to address various problems[6, 7]. For task planning, prompting facilitates LLMs to break down high-level tasks into sub-tasks[22] and formulate grounded plans[23, 24]. ReAct[25] proposes an enhanced integration of reasoning and action, enabling LLMs to provide a valid justification for action and integrating environmental feedback into the reasoning process. BabyAGI, AgentGPT, and AutoGPT also adopt step-by-step thinking, which iteratively generates the next task by using LLMs, providing some solutions for task automation. However, these methods become problematic as an initial error can propagate along an action sequence, leading to a cascade of subsequent errors. Reflexion[26] incorporates a mechanism for decision retraction, asking LLMs to reflect on previous failures to correct their decision-making. HuggingGPT[27] adopts a global planning strategy to obtain the entire sub-task queue within one user query. It is difficult to judge whether iterative or global planning is better since each one has its deficiencies and both of them heavily rely on the ability of LLMs, despite these models not being specifically tailored for task planning. Besides the above LLM-based studies, previous hierarchical agents, such as SEIHAI [28], Juewu-MC [29], GITM [30] often resemble the spirit of task planning.

However, in real-world systems, the high-level tasks are more intricate, and the prompting method without enhancing the intrinsic task-planning ability of LLMs can hardly achieve good performance. Thus, in our work, we adopt a fine-tuning mechanism to the planning dataset, along with well-designed prompts, to maximize the ability of task planning.

4.2 Tool Usage

The initial research in tool learning is limited by the capabilities of traditional deep learning approaches because of their weaknesses in comprehension of tool functionality and user intentions, as well as common sense reasoning abilities. Recently, the advancement of LLM has marked a pivotal juncture in the realm of tool learning. The great abilities of LLMs in common sense cognition and natural language processing attributes furnish indispensable prerequisites for LLMs to comprehend user intentions and effectively employ tools in tackling intricate tasks[31]. Additionally, tool usage can alleviate the inherent limitations of LLMs, encompassing the acquisition of up-to-date information from real-world events, enhanced mathematical computational abilities, and the mitigation of potential hallucinatory phenomena[32].

In the domain of embodied intelligence[33], LLMs directly interact with tangible tools, such as robots, to augment their cognitive abilities, optimize work productivity, and broaden functional capacities.LLM possesses the capability to automatically devise action steps according to user

intentions, facilitating the guidance of robots in task completion[34–36, 24, 37–41], or alternatively, to directly generate underlying code that can be executed by robots[42–45, 9].

In addition to directly influencing the physical real world through interactions with tools, LLM can also utilize software tools such as search engines [46, 47], mobile[48, 49], Microsoft Office [50, 51], calculators[52–54], deep models[55, 56] and other versatile APIs[57–59] to improve model performance or complete complex workflows through flexible control of the software.

However, most of the aforementioned works focus only on specific scenarios, addressing how to choose or use the appropriate tools from a limited set, while agents in real-world scenarios usually have to face various and complex situations, requiring precise selection and usage of the correct tools from an API cloud with massive APIs. Gorilla[60] connects LLMs with massive APIs, which are, nonetheless, not real-world APIs and with poor diversity. ToolAlpaca[17] builds a tool-using corpus containing 3938 tool-use instances from more than 400 real-world tool APIs spanning 50 distinct categories, but this method focuses on smaller language models. ToolLLM[16] provides a novel and high-quality prompt-tuning dataset, ToolBench, which collects 16464 real-world APIs spanning 49 categories from RapidAPI Hub, covering both single-tool and multi-tool scenarios. TaskMatrix.AI[59] uses LLM as a core system and connects with millions of APIs to execute both digital and physical tasks. The methods above are of great assistance to the tool-learning research community.

To augment LLMs with external tools, most recent methods rely on few-shot prompting with the off-the-shelf LLMs[60, 17, 61, 62, 18, 63] , but the existing LLMs are not developed for agentic use cases. FireAct[64] proposes a novel approach to fine-tune LLMs with trajectories from multiple tasks and prompting methods and find LLM-based agents are consistently improved after fine-tuning their backbone. ToolLLM[16] uses SFT based on the proposed ToolBench, to transform LLaMa[65] into ToolLLaMa, which demonstrates a remarkable ability to execute complex instructions and generalize to unseen APIs, and exhibits comparable performance to ChatGPT. Inspired by these, we not only design an API Retriever and Demo Selector to serve as an auto-prompter but also employ fine-tuning techniques to further enhance the performance of our framework so that it can address much more complex tasks in real-world scenarios.

5 Conclusion

In this paper, we present a comprehensive framework designed to augment the capabilities of Large Language Models (LLMs) in complex, real-world scenarios, particularly focusing on task planning and tool usage. Our approach, which integrates the API Retriever, LLM Finetuner, and Demo Selector, has been rigorously tested and validated in various settings. The results demonstrate that fine-tuning LLMs with a curated dataset significantly improves their effectiveness in executing real-world tasks. The API Retriever and Demo Selector components also prove indispensable, particularly in enhancing the model's decision-making accuracy and adaptability. This research not only showcases the potential of LLMs in practical applications but also lays a foundation for future advancements in the field. By addressing the challenges of API diversity and complexity, our framework paves the way for more efficient, and user-centric AI systems, capable of handling real-world scenarios.

Acknowledgements

This work was conducted collaboratively among the authors.

Hangyu Mao and Rui Zhao led the project.

Regarding the implementation and evaluation phase, Yihong Chen, Tianpeng Bao, Guoqing Du, Xiaoru Hu, Shiwei Shi, Jingqing Ruan, Yilun Kong and Bin Zhang performed the experiments and analyzed the data. Hangyu Mao assisted in the analysis of the experimental phenomena and offered constructive suggestions for improvements. Ziyue Li, Xingyu Zeng and Rui Zhao provided invaluable feedback, contributed to the direction of the research. All authors participated in the discussion.

Regarding the manuscript phase, Jingqing Ruan and Yilun Kong organized and wrote main parts of this manuscript. Hangyu Mao provided assistance during the process. Each author read and approved the final manuscript.

The authors would like to thank Feng Zhu, Kun Wang, Yuhang Ran, and colleagues from the product-side for their valuable feedback, discussion, and participation in this project.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[3] OpenAI, "Gpt-4 technical report," 2023.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28 492–28 518.

[6] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[7] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[8] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *arXiv preprint arXiv:2305.01210*, 2023.

[9] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[10] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *arXiv preprint arXiv:2308.11432*, 2023.

[11] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou *et al.*, "The rise and potential of large language model based agents: A survey," *arXiv preprint arXiv:2309.07864*, 2023.

[12] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, G. Du, S. Shi, H. Mao, X. Zeng, and R. Zhao, "Tptu: Task planning and tool usage of large language model-based ai agents," *arXiv preprint arXiv:2308.03427*, 2023.

[13] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, H. Mao, X. Zeng, R. Zhao *et al.*, "Tptu: Task planning and tool usage of large language model-based ai agents," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[14] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[15] M. Henderson, R. Al-Rfou, B. Strope, Y.-H. Sung, L. Lukács, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil, "Efficient natural language response suggestion for smart reply," *arXiv preprint arXiv:1705.00652*, 2017.

[16] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian *et al.*, "Toolllm: Facilitating large language models to master 16000+ real-world apis," *arXiv preprint arXiv:2307.16789*, 2023.

[17] Q. Tang, Z. Deng, H. Lin, X. Han, Q. Liang, and L. Sun, "Toolalpaca: Generalized tool learning for language models with 3000 simulated cases," *arXiv preprint arXiv:2306.05301*, 2023.

[18] M. Li, F. Song, B. Yu, H. Yu, Z. Li, F. Huang, and Y. Li, "Api-bank: A benchmark for tool-augmented llms," *arXiv preprint arXiv:2304.08244*, 2023.

[19] I. Team, "Internlm: A multilingual language model with progressively enhanced capabilities," https://github.com/InternLM/InternLM, 2023.

[20] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," *arXiv preprint arXiv:2104.08786*, 2021.

[21] A. Zeng, M. Liu, R. Lu, B. Wang, X. Liu, Y. Dong, and J. Tang, "Agenttuning: Enabling generalized agent abilities for llms," *arXiv preprint arXiv:2310.12823*, 2023.

[22] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[23] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[24] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

[25] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[26] N. Shinn, F. Cassano, A. Gopinath, K. R. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[27] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface," *arXiv preprint arXiv:2303.17580*, 2023.

[28] H. Mao, C. Wang, X. Hao, Y. Mao, Y. Lu, C. Wu, J. Hao, D. Li, and P. Tang, "Seihai: A sample-efficient hierarchical ai for the minerl competition," in *Distributed Artificial Intelligence: Third International Conference, DAI 2021, Shanghai, China, December 17–18, 2021, Proceedings 3*. Springer, 2022, pp. 38–51.

[29] Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, and W. Yang, "Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning," *arXiv preprint arXiv:2112.04907*, 2021.

[30] X. Zhu, Y. Chen, H. Tian, C. Tao, W. Su, C. Yang, G. Huang, B. Li, L. Lu, X. Wang *et al.*, "Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory," *arXiv preprint arXiv:2305.17144*, 2023.

[31] Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, Y. Huang, C. Xiao, C. Han *et al.*, "Tool learning with foundation models," *arXiv preprint arXiv:2304.08354*, 2023.

[32] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz *et al.*, "Augmented language models: a survey," *arXiv preprint arXiv:2302.07842*, 2023.

[33] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, "A survey of embodied ai: From simulators to research tasks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 2, pp. 230–244, 2022.

[34] W. Zhang, Y. Guo, L. Niu, P. Li, C. Zhang, Z. Wan, J. Yan, F. U. D. Farrukh, and D. Zhang, "Lp-slam: Language-perceptive rgb-d slam system based on large language model," *arXiv preprint arXiv:2303.10089*, 2023.

[35] D. Shah, B. Osiński, S. Levine *et al.*, "Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Conference on Robot Learning*. PMLR, 2023, pp. 492–504.

[36] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*. PMLR, 2023, pp. 287–318.

[37] B. Chen, F. Xia, B. Ichter, K. Rao, K. Gopalakrishnan, M. S. Ryoo, A. Stone, and D. Kappler, "Open-vocabulary queryable scene representations for real world planning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 509–11 522.

[38] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[39] N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu, and K. Ikeuchi, "Chatgpt empowered long-step robot control in various environments: A case application," *arXiv preprint arXiv:2304.03893*, 2023.

[40] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable task planning," *arXiv preprint arXiv:2307.06135*, 2023.

[41] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," *arXiv preprint arXiv:2212.04088*, 2022.

[42] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[43] A. Stone, T. Xiao, Y. Lu, K. Gopalakrishnan, K.-H. Lee, Q. Vuong, P. Wohlhart, B. Zitkovich, F. Xia, C. Finn *et al.*, "Open-world object manipulation using pre-trained vision-language models," *arXiv preprint arXiv:2303.00905*, 2023.

[44] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg *et al.*, "A generalist agent," *arXiv preprint arXiv:2205.06175*, 2022.

[45] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, p. 20, 2023.

[46] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *International conference on machine learning*. PMLR, 2020, pp. 3929–3938.

[47] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*. PMLR, 2022, pp. 2206–2240.

[48] B. Wang, G. Li, and Y. Li, "Enabling conversational interaction with mobile ui using large language models," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–17.

[49] D. Zhang, L. Chen, and K. Yu, "Mobile-env: A universal platform for training and evaluation of mobile interaction," *arXiv preprint arXiv:2305.08144*, 2023.

[50] H. Li, J. Su, Y. Chen, Q. Li, and Z. Zhang, "Sheetcopilot: Bringing software productivity to the next level through large language models," *arXiv preprint arXiv:2305.19308*, 2023.

[51] L. Zha, J. Zhou, L. Li, R. Wang, Q. Huang, S. Yang, J. Yuan, C. Su, X. Li, A. Su *et al.*, "Tablegpt: Towards unifying tables, nature language and commands into one gpt," *arXiv preprint arXiv:2307.08674*, 2023.

[52] Z. Chen, K. Zhou, B. Zhang, Z. Gong, W. X. Zhao, and J.-R. Wen, "Chatcot: Tool-augmented chain-of-thought reasoning on\\chat-based large language models," *arXiv preprint arXiv:2305.14323*, 2023.

[53] A. Parisi, Y. Zhao, and N. Fiedel, "Talm: Tool augmented language models," *arXiv preprint arXiv:2205.12255*, 2022.

[54] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[55] T. Gupta and A. Kembhavi, "Visual programming: Compositional visual reasoning without training," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14 953–14 962.

[56] L. Chen, B. Li, S. Shen, J. Yang, C. Li, K. Keutzer, T. Darrell, and Z. Liu, "Language models are visual reasoning coordinators," in *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023.

[57] P. Lu, B. Peng, H. Cheng, M. Galley, K.-W. Chang, Y. N. Wu, S.-C. Zhu, and J. Gao, "Chameleon: Plug-and-play compositional reasoning with large language models," *arXiv preprint arXiv:2304.09842*, 2023.

[58] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen, "Critic: Large language models can self-correct with tool-interactive critiquing," *arXiv preprint arXiv:2305.11738*, 2023.

[59] Y. Liang, C. Wu, T. Song, W. Wu, Y. Xia, Y. Liu, Y. Ou, S. Lu, L. Ji, S. Mao *et al.*, "Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis," *arXiv preprint arXiv:2303.16434*, 2023.

[60] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," *arXiv preprint arXiv:2305.15334*, 2023.

[61] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *arXiv preprint arXiv:2305.10601*, 2023.

[62] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[63] Q. Xu, F. Hong, B. Li, C. Hu, Z. Chen, and J. Zhang, "On the tool manipulation capability of open-source large language models," *arXiv preprint arXiv:2305.16504*, 2023.

[64] B. Chen, C. Shu, E. Shareghi, N. Collier, K. Narasimhan, and S. Yao, "Fireact: Toward language agent fine-tuning," *arXiv preprint arXiv:2310.05915*, 2023.

[65] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.