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

Yilun Kong[12†‡] **Jingqing Ruan**[234†‡] **Yihong Chen**[12†‡] **Bin Zhang**[234†‡]
Tianpeng Bao[2†] **Shiwei Shi**[2†] **Guoqing Du**[2†] **Xiaoru Hu**[2†]
Hangyu Mao[2†✉] **Ziyue Li**[5✉] **Xingyu Zeng**[2] **Rui Zhao**[26] **Xueqian Wang**[1]
[1]Shenzhen International Graduate School, Tsinghua University [2]SenseTime Research
[3]Institute of Automation, Chinese Academy of Sciences
[4]School of Artificial Intelligence, University of Chinese Academy of Sciences
[5]University of Cologne, Germany [6]Qingyuan Research Institute, Shanghai Jiaotong University

Abstract

Large Language Models (LLMs) have demonstrated proficiency in addressing tasks that necessitate a combination of task planning and the usage of external tools, such as weather and calculator APIs. However, real-world industrial systems present prevalent challenges in task planning and tool usage: numerous APIs in the real system make it intricate to invoke the appropriate one, while the inherent limitations of LLMs pose challenges in orchestrating an accurate sub-task sequence and API-calling order. This paper introduces a comprehensive framework aimed at enhancing the Task Planning and Tool Usage (TPTU) abilities of LLM-based agents in industry. Our framework comprises three key components designed to address these challenges: (1) the *API Retriever* selects the most pertinent APIs among the extensive API set; (2) the *Demo Selector* retrieves task-level demonstrations, which is further used for in-context learning to aid LLMs in accurately decomposing subtasks and effectively invoking hard-to-distinguish APIs; (3) *LLM Finetuner* tunes a base LLM to enhance its capability for task planning and API calling . We validate our methods using a real-world industry system and an open-sourced academic dataset, demonstrating the efficacy of each individual component as well as the integrated framework. The code is available at here.

1 Introduction

Large language models (LLMs) have exhibited remarkable prowess in various domains of natural language processing (NLP) (Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023b), encompassing language understanding (Devlin et al., 2018; Radford et al., 2023), reasoning (Wei et al., 2022; Kojima et al., 2022), and program synthesis (Liu et al., 2023; Liang et al., 2023a).

However, leveraging LLMs for complex tasks presents formidable challenges. On the one hand, LLMs inherently exhibit limitations in their capabilities, struggling with logical problem-solving, such as mathematics, and facing the risk of stored knowledge quickly becoming outdated as the world evolves. Instructing LLMs to utilize external tools including calculators, calendars, or search engines can prevent them from generating inaccurate information and aid them in effectively addressing problems. On the other hand, integrating these tools 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 (Wang et al., 2023c; Cheng et al., 2024; Hua et al., 2024), such as AutoGPT (Significant Gravitas, 2023), BabyAGI (yoheinakajima, 2023), and ChatGPT-plugins (OpenAI, 2023a), have made advancements by employing LLMs as central controllers. These endeavors automatically decompose user queries into sub-tasks, execute low-level tool (i.e., 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 industry 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 (i.e., general LLMs without finetuning on these tasks) often struggle to correctly plan sub-task orders and API-calling sequences for such tasks. Thirdly, beyond the sheer quantity of APIs, a more substantial challenge is that real systems are primarily designed around a core purpose, resulting in the fact

[†]These authors contribute equally to this work.
[‡]These authors work as interns at SenseTime Research.
[✉]The corresponding authors:
`hy.mao@pku.edu.cn; zlibn@wiso.uni-koeln.de`

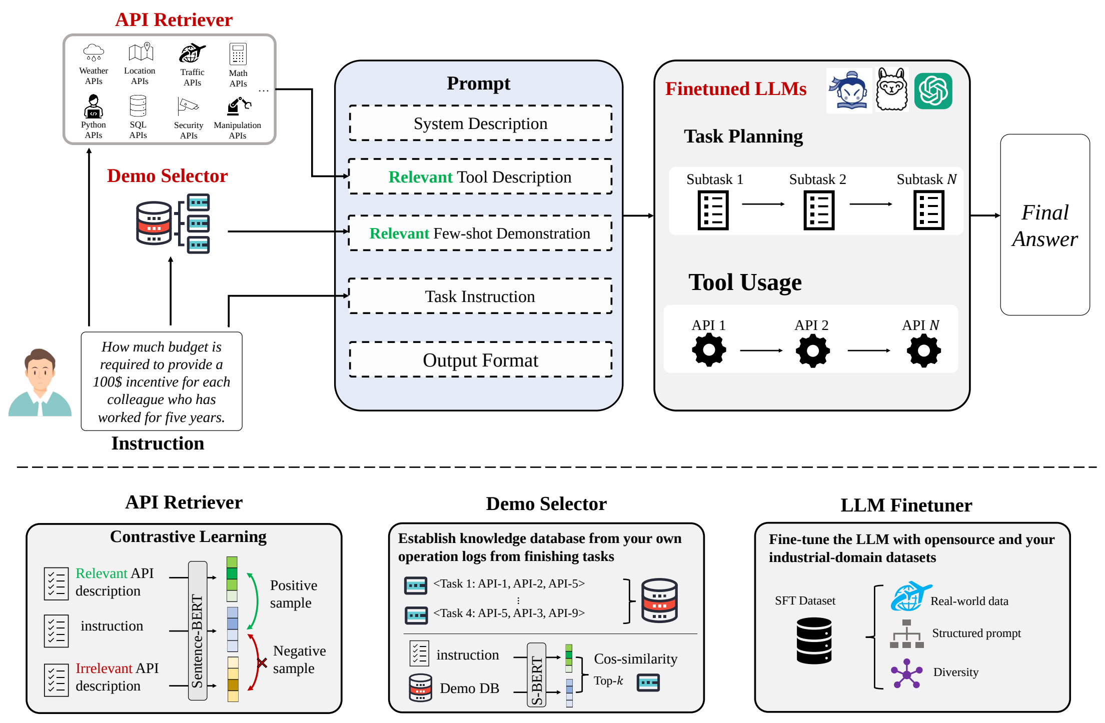


Figure 1: The proposed framework: Once receiving user's instruction, the *API Retriever* and *Demo Selector* will get the relevant APIs and Demos first, which will be inserted as the crucial segment in the prompt. The *LLM Finetuner* processes the prompt, decomposes the task into several subtasks and their corresponding APIs, and iteratively executes the API for each subtask. Specifically, (1) API Retriever is based on contrastive learning; (2) Demo Selector is based on a self-established industrial-specific knowledge database; (3) LLM Finetuner is fine-tuning the LLMs with the dataset from real-world, structure-prompted, and diversity.

that certain APIs may overlap and exhibit similar semantics and functionality, while different in usage. For instance, variations in required input parameters create difficulties in distinguishing and invoking these APIs for LLMs and even humans. How to address these issues could be the critical step for LLM-based agents towards omniscience and omnipotence and being implemented in real-world industry scenarios.

In this paper, we propose a framework to improve the **T**ask **P**lanning and **T**ool **U**sage (TPTU) abilities of LLM-based agents in the real-world systems, which is composed of three key components to address the above three challenges: (1) **API Retriever** that is based on contrastive learning, to accurately recall 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) **Demo Selector** that is based on our self-established knowledge base using the logs when performing tasks, it 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. (3) **LLM Finetuner** tunes a base LLM with a dataset of three essential criteria, i.e., real-world, structured prompt, and diversity, so that the finetuned LLM can be more capable of task planning and API calls, especially for domain-specific tasks. Our main contributions can be summarized as follows:

- We identify three practical challenges LLM-based agents face in task planning and tool usage within real-world industry scenarios.
- Facing the three challenges, we propose an advanced framework composed of three key components: API Retriever, LLM Finetuner, and Demo Selector. In each component, key technical designs are equipped to further enhance its performance: contrastive learning to boost API retrieval, self-knowledge based to get better-adapting few-shot demos for your industrial domain, and a real-world-based, structured-prompted, and diverse data to fine-tune the base LLM for better TPTU.

- Extensive experiments in real-world industrial systems demonstrate the effectiveness our framework. We also validate our methods with open-sourced academic datasets.

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), which are the cornerstone of the agent's abilities. In this paper, Task Planning entails generating a step-by-step subtask sequences for the complex task, while Tool Usage requires the agent to select appropriate APIs and execute them correctly to obtain the answer.

2.1 Framework Overview

Our comprehensive framework is engineered to enhance the capabilities of LLMs in TPTU within complex real-world systems, facilitating synergistic collaboration between these two abilities. It has three pivotal components, as in Figure 1. When receiving user instructions, it initiates the process by acquiring pertinent APIs and demonstrations through the API Retriever and Demo Selector. These API descriptions and demos constitute a crucial segment of the prompt, which is then input into the fine-tuned LLM. The LLM processes the instruction, leveraging the obtained APIs and demonstrations to derive subtasks and their corresponding API calls. Subsequently, it iteratively executes the API for each subtask, obtaining sub-results and ultimately achieving the complete result, satisfying the strict requirements for API call order in real industrial scenarios. Details of input prompt and output format are in Figure 2 and Appendix B.1.

2.2 API Retriever

In real-world systems, the massive number of APIs poses severe challenges for LLMs. The token length limitations inherent to LLMs hinder the inclusion of all API descriptions in the model's prompt, while excessive task-irrelevant API information impedes planning and answer generation.

To surmount these challenges, we develop an API Retriever trained to select APIs most relevant to the overall task. These selected APIs are not only necessary for solving the overall task, but also contribute to enhancing the LLM's comprehension of the task at hand. This, in turn, facilitates more precise segmentation of sub-tasks and the accurate execution of API calls.

The module is a dual-stream architecture employing two Sentence-BERT models (Reimers and Gurevych, 2019) to obtain semantic embeddings of the instruction and API descriptions, separately. It selects the API description closest to the instruction in our trained semantic space. We use the Multiple Negatives Ranking Loss (MNR Loss) (Henderson et al., 2017) to explicitly contrast the positive pair (an instruction with the relevant API) against multiple negative pairs (with irrelevant API), minimizing the distance between the embeddings of correct instruction-API pairs while maximizing the distance between the embeddings of incorrect pairs, which can be formulated as follows:

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

K denotes the batch size, s_i indicates the sentence embedding of instruction i, while s_i^+ and s_j^- represent the embeddings of API descriptions which form the positive and negative pairs corresponding to s_i, respectively. $sim(\cdot)$ is the cosine similarity.

The effectiveness of the API Retriever is also grounded in a meticulous data collection process. We compile a comprehensive set of APIs from diverse external tool services. To ensure our system grasps the relevance of different APIs to various user instructions, we implement a annotation process. Human experts and LLMs analyze complex user instructions to identify and annotate the APIs necessary for resolving these instructions, which forms a robust foundation for the API Retriever.

2.3 Demo Selector

The Demo Selector, serving as an in-context learning method to provide few-shot demonstrations, plays a crucial role in instructing LLMs to distinguish between potentially confusing APIs [1], execute APIs accurately and disassemble complex tasks. We establish a knowledge database from real-world industry scenarios. This knowledge database can be easily compiled, as the routine operations of industry systems naturally generate numerous operationally correct records derived from real-world situations. These records, closely aligned with user

[1]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.

instructions, constitute the foundation of our knowledge database. Thus, the knowledge database is highly industrial-specific.

The Demo Selector consists of a pre-trained Sentence-BERT and uses an embedding search mechanism to select appropriate demonstrations from the knowledge database. In contrast to the API Retriever, this module does not require fine-tuning with new training data, as the API Retriever matches user instructions with API descriptions, while the Demo Selector matches user instructions with task instructions, which inherently share the same semantic space. The detailed processes are:

1. **Embedding Generation**. Initially, the user's query Q and demonstrations d_i from the knowledge database D are transformed into semantic embeddings $emb(Q)$ and $emb(d_i)$.
2. **Top-k Task-Level Demos**. In candidates with similarity exceeding a pre-defined threshold $sim(emb(Q), emb(d_i)) > \Delta$, we select the top-$k$ demonstrations based on their similarity scores. These are regarded as task-level demonstrations as they are closely related to the specific task at hand.
3. **Fallback to API-Level Demos**. If there only exists n demonstrations with similarity exceeding the threshold, where $n < k$, the process degrades to selecting API-level demonstrations from the API collection. The module chooses $k - n$ API-level demonstrations (i.e., application examples of a specific API) based on the order of API description.

2.4 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. Fine-tuning LLMs on downstream tasks is a prevailing practice to refine their proficiency in addressing specific challenges in these domains. Since the ubiquity and satisfactory performance of existing fine-tuning methods, such as SFT (Ouyang et al., 2022) and LoRA (Hu et al., 2021), 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*. Compared to sophisticated fine-tuning methods, even non-technical personnel can help construct appropriate training data, making it more suitable for industrial applications.

Specifically, we employ SFT to fine-tune our model and meticulously construct a well-designed dataset with the following characteristics: (1) **Real-world Data:** To accurately mirror real-world scenarios, the dataset is constructed by carefully selecting genuine cases, so that the fine-tuned LLMs can align with the real data distribution in practical use. (2) **Structured Prompt:** The prompts in the dataset are augmented with several key elements, including the system descriptions, API descriptions and demonstrations, which enables the model to generate responses that not only semantically match the input query but also closely align with the functional scope of the available APIs. (3) **Diversity:** To further capture real-world situations, we expand the diversity of the dataset, including prompt diversity, user instruction diversity, and output diversity. Both prompt and instruction diversity are crucial for enabling the LLM to navigate the API space with greater precision, particularly when haddling complex, multi-faceted user requests. For output diversity, the incorporation of various single-step and multi-step API interactions serves to not only solidify the foundational understanding of API functionalities, but also expose the LLM to more complex sequences of operations commonly encountered in practice. More details of our dataset's features can be seen in Appendix B.2.

3 Experiments

We present comprehensive experiments to rigorously evaluate the efficacy of our proposed framework. Experiments are structured to assess the framework's performance in both a real-world scenario and an open-source academic challenge to analysis our framework's generalization.

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 and require multiple tool invocations. It entails choosing the most suitable API from a few dozen APIs, which are closely related in semantics but differ in usage, and orchestrating the correct order for these API calls, enabling bootstrapping solutions for complex problems. Therefore, we construct

Table 1: Comparison between our real-world industrial dataset and notable open-source datasets. Our method focuses on real industrial datasets where the semantics or usage of APIs are relatively similar. Each problem requires more APIs to be solved, and the order of API sequential execution is strictly constrained. Additionally, we also focus on the method's generalizability on open-source academic datasets.

Resource	ToolBench (Qin et al., 2023b)	APIBench (Patil et al., 2023)	API-Bank (Li et al., 2023b)	Ours
Real-world API	✓	✗	✓	✓
Real-world Query	✗	✗	✗	✓
Multi-tool Scenario	✓	✗	✗	✓
Multi-step Reasoning	✓	✗	✓	✓
Manual Construction	✗	✗	✗	✓
# APIs	16464	1645	53	45
# Instances	12657	17002	274	760
Avg. # API Calls	2.94	1	2.76	**3.50**

a specialized dataset comprising 45 core APIs [2] utilized in industry application, based on a real system. To align with real-world scenarios, the dataset includes 390 single-call samples and 370 multi-call samples. The multi-call samples involve up to 9 API calls, with an average of 3.5 API calls across the entire dataset, which is larger than many open-source datasets. We meticulously selected real-world instructions, incorporating simple questions with fewer than 10 words and challenging questions with more than 100 words. 760 instances are used for training, while for testing, we employ additional problems that are collected from the industrial system in real-time, which are completely distinct from those in the dataset. The detailed statistics are shown in Table 1, and the examples of simple and challenging real-world industry questions are provided in Appendix C.1.

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 (Qin et al., 2023b), which also closely resembles real-world applications. It contains 16000+ real-world APIs spanning 49 application categories.

3.2 Baselines

In the real-world scenario, we select both closed-source and open-source LLMs as baselines, including GPT-3.5, Claude (Anthropic, 2023), and InternLM (Team, 2023). In the open-source scenario, our baselines include GPT-3.5 and ToolLLaMA, which tailors for ToolBench. More related works can be found in Appendix A.

[2] During testing, we combine the 45 core APIs with thousands of irrelevant APIs, and our API Retriever can easily filter out these unrelated APIs, which shows that real-world challenges may not align with the academic notion that the increasing number of APIs makes the problem more challenging. Thus, we only collect the core APIs in our dataset.

Table 2: Performance comparison on real-world scenario, where GTA and AR denote using ground truth APIs and APIs selected by API Retriever respectively, and DS represents utilizing Demo Selector for in-context learning.

Approaches	Accuracy
GPT-3.5 + GTA	70.0%
Claude + GTA	86.7%
InternLM-ft + AR + DS (ours)	**96.67%**
InternLM	16.70%
InternLM + GTA	38.89%
InternLM + AR	43.33%
InternLM-ft + GTA	80.48%
InternLM-ft + AR	80.34%
InternLM + GTA + DS	95.55%
InternLM-ft + GTA + DS	<u>100%</u>

3.3 Main Experiment on Real-world Scenario

In our real-world dataset, we conduct experiments to assess the efficacy of all proposed modules in our framework. We employ the LLM Finetuner on the open-source InternLM to underscore the importance of fine-tuning in the TPTU framework.

Main Results As shown at the top of Table 2, our method significantly outperforms the baselines for TPTU. The remarkable performance is attained through the integration of the finetuned InternLM (InternLM-ft) with both the API Retriever and Demo Selector, achieving an impressive accuracy of 96.67%, a level sufficient for practical application in real-world commercial scenarios.

API Retriever We utilize the top-5 APIs recommended by the API Retriever. The results show that: **(1) Employing API Retriever can achieve great performance.** Utilizing API Retriever yields comparable or better results than using ground-truth APIs, and significantly improves the performance over not containing API in the prompt (i.e., 43.33% v.s. 16.70%). **(2) When the model is strong (w. finetuning), API Retriever can deliver comparable accuracy** (i.e., 80.34% v.s. 80.48% and 96.67% v.s. 100%). The finetuned LLM has the ability to better understand the prompt and decompose tasks, using ground-truth APIs can avoid the slight errors introduced by the API Retriever. **(3) When the model is weak (w/o finetuning), API Retriever can yield better results** (i.e., 43.33% v.s. 38.89%) **by reordering the API sequence and enriching**

the API's diversity in the prompt. As the diversity and relative position of APIs within the prompt can affect the LLM's interpretation of the context and the relationships between different APIs, ultimately influencing its output (Lu et al., 2021). Thus, this module might be a promising approach to automatically retrieve the appropriate APIs as our methods demonstrated.

Demo Selector We directly compare the difference between simply using the API usage examples and utilizing the top-5 demonstrations acquired through Demo Selector as in-context demonstrations. During selecting demonstrations, we set the similarity threshold as 0.8. The results show that **Demo Selector also has a substantial impact in each set of ablation experiments** (i.e., 95.55% v.s. 38.89% and 100% v.s. 80.48%), due to its ability to provide context-rich examples that guide the LLM in making more informed decisions.

LLM Finetuner Regarding the benefits of fine-tuning, the results clearly demonstrate its advantages. **In all experimental configurations, the accuracy of the InternLM-ft is significantly higher than that of the base one**. Specifically, in the two experimental setups without DS, fine-tuning achieves significant performance gains (i.e., 80.48% v.s. 38.89%, 80.34% v.s. 43.33%), allowing the model to plan and execute API calls without contextual demonstrations. In experiments with DS, where the base model can solve problems using demonstrations, fine-tuning also further enhanced its performance (100% v.s. 95.55%). The fine-tuning process tailors the LLM more closely to the specifics of the real-world industry task. It enhances the model's understanding of the context, leading to more accurate and contextually appropriate API calls.

In conclusion, our experiments in a real-world setting validate the efficacy of our proposed framework, highlighting the importance of each component and demonstrating our approach is applicable in practical applications. Cases of our method's input and output sequences on real-world industry datasets are presented in Appendix C.4.

3.4 Experiments on API Retriever

To further elucidate the factors contributing to the effectiveness of the API Retriever, this section focuses specifically on its ability to select correct APIs as well as the characteristics of the training dataset for API Retriever.

One key factor for the strong performance of API Retriever in the entire framework is its **high precision in recalling the correct APIs**, which ensures minimal deviation between the retrieved APIs and the ground truth. Moreover, the ranked order of retrieved APIs, based on similarity, further enhances the overall performance. In particular, this module achieves a Recall@5 precision of 84.64% and Recall@10 precision of 98.47% in the combination of core and irrelevant APIs. After subsequent experiments, we ultimately adopt the strategy of recalling the top-5 APIs.

To safeguard the API Retriever from overfitting to the real-world dataset and enhance its generalizability, **we employ both the real-world dataset and ToolBench for training**. This comprehensive training set encompasses a total of 8330 data points, ensuring a more robust and adaptable performance.

Table 3: The performance, based on GTA, of InternLM fine-tuned on datasets with different feature ablations. RD denotes Real-world Data.

Training Dataset	Accuracy
w/o finetuning	20.0%
w. finetuning	
+ Real-world Data (RD)	0%
+ RD + Structured Prompt	26.7%
+ RD + Structured Prompt + Diversity	80.5%

3.5 Experiments on LLM Finetuner

As shown in Section 2.4, we focus on developing a meticulous dataset to enhance the finetuning process in real-world TPTU. In this section, we conduct ablation experiments on the dataset's characteristics separately to demonstrate the crucial role of our designed features. The results, presented in Table 3, indicate that **dataset with all the characteristics significantly improves the effect of fine-tuning**, achieving an accuracy of 80.5%. The lack of diversity in the dataset results in a notable decline in model performance, with a fine-tuning accuracy of only 26.7%. This is because the model may tend to rotely memorize the API call solutions for different instructions. **Fine-tuning solely on raw real-world data can backfire**, for the model may overfit to specific issues within the training set due to the constraints imposed by the limited quantity of genuine data.

We also compared the performance of SFT and LoRA, which is shown in Appendix C.2. Results show that using SFT achieves better performance than LoRA.

3.6 Main Experiment on Open-source Dataset

Due to the length limitation, the main results on the open-source dataset are shown in Appendix C.3. To highlight, our approach achieves an accuracy of 87.6%, outperforming two strong baselines, i.e., GPT-3.5 and ToolLLaMa. Both API Retriever and fine-tuning significantly contribute to the overall performance. The main reason for the performance decline compared to the industry dataset is that we do not construct a dedicate knowledge database nor introduce the Demo Selector, thus lacking demonstrations of the overall task.

4 Conclusion

In this paper, we present a comprehensive framework to augment the capabilities of LLMs in real-world scenarios, with a specific focus on **T**ask **P**lanning and **T**ool **U**sage (TPTU). Our approach, which integrates API Retriever, LLM Finetuner, and Demo Selector, has been validated in both a real-world scenario and an open-source setting. The results demonstrate that fine-tuning LLMs with a curated dataset can significantly improve their effectiveness in executing real-world tasks. The API Retriever and Demo Selector also prove indispensable, particularly in improving the model's decision-making accuracy and adaptability. This research not only highlights the potential of LLMs in practical applications but also establishes a foundation for future advancements in this field.

5 Ethics Statement

The examples provided in this paper, including the surveillance and relationship analysis scenarios, are based on a simulated detective game from the real-world, i.e., "The Mystery Solver", designed to evaluate the technical capabilities of the AI system in a fictional context. This test environment mimics investigative tasks in a controlled, gamified scenario where no real individuals or personal data are involved. The purpose is purely academic and aimed at improving AI's ability to process structured queries within a safe and ethical framework.

No real-world surveillance or relationship analysis was conducted. Furthermore, should any real-world applications of this technology be considered, they would be subject to strict ethical guidelines, legal regulations, and the protection of privacy through informed consent.

Last but not the least, we recognize the potential societal impacts of AI technologies, particularly those involving sensitive tasks such as surveillance or personal data analysis. Our work is guided by a commitment to ensuring that AI is developed and applied ethically, with a focus on transparency, fairness, and respect for individual rights.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

AI Anthropic. 2023. Introducing claude.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. 2022. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023a. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Boyuan Chen, Fei Xia, Brian Ichter, Kanishka Rao, Keerthana Gopalakrishnan, Michael S Ryoo, Austin Stone, and Daniel Kappler. 2023b. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11509–11522. IEEE.

Liangyu Chen, Bo Li, Sheng Shen, Jingkang Yang, Chunyuan Li, Kurt Keutzer, Trevor Darrell, and Ziwei Liu. 2023c. Language models are visual reasoning coordinators. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji-Rong Wen. 2023d. Chatcot: Tool-augmented chain-of-thought reasoning on\\chat-based large language models. *arXiv preprint arXiv:2305.14323*.

Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, et al. 2024. Exploring large language model based intelligent agents: Definitions, methods, and prospects. *arXiv preprint arXiv:2401.03428*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. 2022. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Wenyue Hua, Xianjun Yang, Zelong Li, Cheng Wei, and Yongfeng Zhang. 2024. Trustagent: Towards safe and trustworthy llm-based agents through agent constitution. *arXiv preprint arXiv:2402.01586*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Yilun Kong, Jingqing Ruan, YiHong Chen, Bin Zhang, Tianpeng Bao, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, Xueqian Wang, et al. 2024. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. 2023a. Sheetcopilot: Bringing software productivity to the next level through large language models. *arXiv preprint arXiv:2305.19308*.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. Api-bank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, et al. 2024. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023a. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023b. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. 2021. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:2112.04907*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*.

Hangyu Mao, Chao Wang, Xiaotian Hao, Yihuan Mao, Yiming Lu, Chengjie Wu, Jianye Hao, Dong Li, and Pingzhong Tang. 2022. 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*, pages 38–51. Springer.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

OpenAI. 2023a. ChatGPT-Plugin.

OpenAI. 2023b. Gpt-4 technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. 2023. Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. *arXiv preprint arXiv:2307.06135*.

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. 2022. A generalist agent. *arXiv preprint arXiv:2205.06175*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*.

Dhruv Shah, Błażej Osiński, Sergey Levine, et al. 2023. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on Robot Learning*, pages 492–504. PMLR.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Significant Gravitas. 2023. AutoGPT.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2022. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088*.

Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, et al. 2023. Open-world object manipulation using pre-trained vision-language models. *arXiv preprint arXiv:2303.00905*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

InternLM Team. 2023. Internlm: A multilingual language model with progressively enhanced capabilities. `https://github.com/InternLM/InternLM`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res*, 2:20.

Naoki Wake, Atsushi Kanehira, Kazuhiro Sasabuchi, Jun Takamatsu, and Katsushi Ikeuchi. 2023. Chatgpt empowered long-step robot control in various environments: A case application. *arXiv preprint arXiv:2304.03893*.

Bryan Wang, Gang Li, and Yang Li. 2023a. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–17.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023b. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023c. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

yoheinakajima. 2023. BabyAGI.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*.

Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, et al. 2023. Tablegpt: Towards unifying tables, nature language and commands into one gpt. *arXiv preprint arXiv:2307.08674*.

Bin Zhang, Hangyu Mao, Jingqing Ruan, Ying Wen, Yang Li, Shao Zhang, Zhiwei Xu, Dapeng Li, Ziyue Li, Rui Zhao, et al. 2024a. Controlling large language model-based agents for large-scale decision-making: An actor-critic approach. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024b. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *arXiv preprint arXiv:2403.02951*.

Danyang Zhang, Lu Chen, and Kai Yu. 2023a. Mobile-env: A universal platform for training and evaluation of mobile interaction. *arXiv preprint arXiv:2305.08144*.

Weiyi Zhang, Yushi Guo, Liting Niu, Peijun Li, Chun Zhang, Zeyu Wan, Jiaxiang Yan, Fasih Ud Din Farrukh, and Debing Zhang. 2023b. Lp-slam: Language-perceptive rgb-d slam system based on large language model. *arXiv preprint arXiv:2303.10089*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. 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*.

A Related Works

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(Kong et al., 2024; Zhang et al., 2024a,b; Li et al., 2024). 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(Ruan et al., 2023) proposes a complete framework that enhances the agent's ability in task planning and tool utilization for addressing complex tasks. AgentTuning(Zeng et al., 2023) 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 (Wang et al., 2023c; Xi et al., 2023).

A.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(Wei et al., 2022; Kojima et al., 2022). For task planning, prompting facilitates LLMs to break down high-level tasks into sub-tasks(Huang et al., 2022a) and formulate grounded plans(Ahn et al., 2022; Huang et al., 2022b). ReAct(Yao et al., 2022) 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(Shinn et al., 2023) incorporates a mechanism for decision retraction, asking LLMs to reflect on previous failures to correct their decision-making. HuggingGPT(Shen et al., 2023) 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 (Mao et al., 2022), Juewu-MC (Lin et al., 2021), GITM (Zhu et al., 2023) often resemble the spirit of task planning.

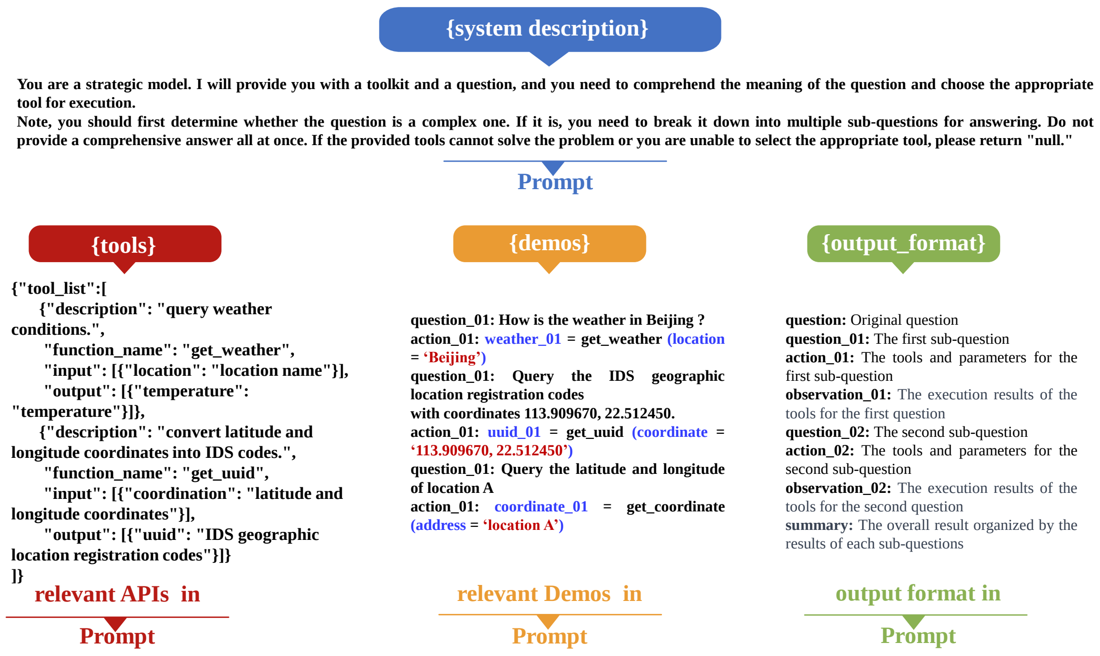
However, in real-world systems, the high-level tasks are more intricate, and the prompting method without enhancing the intrinsic task-planning ability of LLMs can hardly achieve good performance. Thus, in our work, we adopt a fine-tuning mechanism to the planning dataset, along with well-designed prompts,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

A.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(Qin et al., 2023a). 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(Mialon et al., 2023).

In the domain of embodied intelligence(Duan et al., 2022), 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(Zhang et al., 2023b; Shah et al., 2023; Brohan et al., 2023; Huang et al., 2022b; Chen et al., 2023b; Driess et al., 2023; Wake et al., 2023; Rana et al., 2023; Song et al., 2022), or alternatively, to directly generate underlying code that can be executed by robots(Brohan et al., 2022; Stone et al., 2023; Reed et al., 2022; Vemprala et al., 2023; Liang et al., 2023a).

In addition to directly influencing the physical real world through interactions with tools, LLM can also utilize software tools such as search engines (Guu et al., 2020; Borgeaud et al., 2022), mobile(Wang et al., 2023a; Zhang et al., 2023a), Microsoft Office (Li et al., 2023a; Zha et al., 2023), calculators(Chen et al., 2023d; Parisi et al., 2022; Cobbe et al., 2021), deep models(Gupta and Kembhavi, 2023; Chen et al., 2023c) and other versatile APIs(Lu et al., 2023; Gou et al., 2023; Liang et al., 2023b) to improve model performance or complete complex workflows through flexible control of the



You are a strategic model. I will provide you with a toolkit and a question, and you need to comprehend the meaning of the question and choose the appropriate tool for execution.
Note, you should first determine whether the question is a complex one. If it is, you need to break it down into multiple sub-questions for answering. Do not provide a comprehensive answer all at once. If the provided tools cannot solve the problem or you are unable to select the appropriate tool, please return "null."

```
{"tool_list":[
    {"description": "query weather conditions.",
     "function_name": "get_weather",
     "input": [{"location": "location name"}],
     "output": [{"temperature": "temperature"}]},
    {"description": "convert latitude and longitude coordinates into IDS codes.",
     "function_name": "get_uuid",
     "input": [{"coordination": "latitude and longitude coordinates"}],
     "output": [{"uuid": "IDS geographic location registration codes"}]}
]}
```

question_01: How is the weather in Beijing ?
action_01: weather_01 = get_weather (location = 'Beijing')
question_01: Query the IDS geographic location registration codes with coordinates 113.909670, 22.512450.
action_01: uuid_01 = get_uuid (coordinate = '113.909670, 22.512450')
question_01: Query the latitude and longitude of location A
action_01: coordinate_01 = get_coordinate (address = 'location A')

question: Original question
question_01: The first sub-question
action_01: The tools and parameters for the first sub-question
observation_01: The execution results of the tools for the first question
question_02: The second sub-question
action_02: The tools and parameters for the second sub-question
observation_02: The execution results of the tools for the second question
summary: The overall result organized by the results of each sub-questions

Figure 2: Demonstrations of the specific formats of each component in the input prompt and output solutions.

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(Patil et al., 2023) connects LLMs with massive APIs, which are, nonetheless, not real-world APIs and with poor diversity. ToolAlpaca(Tang et al., 2023) 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(Qin et al., 2023b) 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(Liang et al., 2023b) 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(Patil et al., 2023; Tang et al., 2023; Yao et al., 2023; Wang et al., 2023b; Li et al., 2023b; Xu et al., 2023) , but the existing LLMs are not developed for agentic use cases. FireAct(Chen et al., 2023a) 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(Qin et al., 2023b) uses SFT based on the proposed ToolBench, to transform LLaMa(Touvron et al., 2023) 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.

B More details of the method

B.1 Specific components in the prompt

To better understand the problems that our framework addresses, we display the prompt constructed from the input query and the framework's output format in Figure 2. These components form the complete prompt shown in Figure 1, which is then input into the fine-tuned LLM to obtain the output solutions shown in Figure 2.

Real-world Data	Structured Prompt	Diversity
"**Input**": "There is a... icon in the Person File Search — File Details — Personnel File Details interface. Click to pop up the [Historical Modification Record] entrance." "**Output**": "Holographic Profile — Personal Profile Details."	"**System description**" : " You are a system decision-making expert. Below I will provide a complex issue about public security... " …… "**Input**": "There is a... icon in the Person File Search — File Details — Personnel File Details interface. Click to pop up the [Historical Modification Record] entrance." "**Output**": "Holographic Profile — Personal Profile Details."	"**System description**": "You are an expert in system decision-making. I will now present a complex matter related to public security... " …… "**Input**": "How to create a new portrait library and then upload face photos?" "**Output**": "1. Library Management — Creation; 2. Library Assistant — Upload."

Figure 3: Demonstrations for the features in our dataset.

B.2 Demonstrations of LLM Finetuner dataset

To ensure readers can clearly understand each characteristics in the dataset, we provide a demonstration for each one, as shown in Figure 3. Each design of the features 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. 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.

In the details of diversity features, for the prompt diversity, we randomly shuffle API orders and add irrelevant APIs to decrease the risk of over-fitting; 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 to make LLMs more robust to different user instructions during inference. For output diversity, besides simple single-step API interactions, which solidify the foundational understanding of API functionalities, we meticulously select and construct multi-step API calls, which introduce the LLM to more complex sequences of operations that are commonly encountered in practice.

C Supplemental experiment details

C.1 Examples of the questions in our real-world dataset

The examples provided in this paper, including the surveillance and relationship analysis scenarios, are based on a simulated detective game from the real-world, i.e., "The Mystery Solver", designed to evaluate the technical capabilities of the AI system in a fictional context. This test environment mimics investigative tasks in a controlled, gamified scenario where no real individuals or personal data are involved. In order to facilitate a better understanding of the real-world instructions in our dataset, and without compromising the confidentiality of proprietary datasets, we present a simple question and a complex question for illustration.

The following is the simple questions in our dataset:

- Implementing surveillance on a group of individuals.

While in real-world industry scenarios, there are numerous complex problems, with lengths potentially exceeding 100 and comprising multiple subproblems. To enhance our framework's ability to address these issues, we have carefully selected a variety of challenging problems. The following serves as example of these challenging problems:

- In the routine investigation work of criminal detectives, they typically conduct preliminary analysis to identify a category of suspects. Subsequently, they need to track, inspect, and control the identified targets. During this phase, they analyze associated clues and information related to the suspects, ultimately formulating a comprehensive arrest

Table 4: Performance comparison between SFT and LoRA.

Methods	Accuracy
SFT	80.5%
LoRA	26.7%
LoRA (Convergence)	40%

plan. What modules do you think I would be involved in, and could you list the modules in order?

C.2 Comparison experiments between SFT and LoRA

We further discuss the performance of specific fine-tuning methods. We compare the performance of SFT and LoRA, with results displayed in Table 4. It can be seen that when LoRA and SFT are trained for the same number of epochs, LoRA's performance is significantly lower. Even when increasing the number of training epochs for LoRA until the loss converges to the same level as SFT, its performance still lags behind the SFT model. Additionally, analysis of the outputs of models from different training methods reveals that the LoRA fine-tuned LLM struggles to overcome the base LLM's tendency to generate repetitive responses and perform redundant result analysis, while the SFT model is capable of producing concise outputs as required by the prompt. Therefore, under conditions where computational resources are sufficient, using SFT achieves better performance.

C.3 Main 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 knowledge database for this context do not contribute to addressing real-world industry problems. Therefore, the assessment of the Demo Selector is omitted in this scenario and we simply use the API-level demonstrations as in-context examples.

Initially, we have trained the API Retriever on the integration of our dataset and ToolBench, enabling it to generalize in the open-source scenario. In particular, this module achieves a Recall@5 precision of 77.92% and Recall@10 precision of 87.54%, which fall short compared with the results in the industry scenario, posing a challenge for subsequent performance evaluations.

Table 5: Performance comparison on Open-source scenario, where "ft" denotes fine-tuned, "GTA" denotes using ground truth API set, "AR" denotes using API Retriever to select APIs.

Approaches	Accuracy
ToolLLaMA + GTA	74.3%
ChatGPT + GTA	83.6%
InternLM + GTA	76.67%
InternLM + AR	53.3%
InternLM-ft + AR (ours)	87.6%

Main Results As shown in Table 5, our framework (InternLM-ft + AR) outperforms two baseline algorithms, achieving an accuracy of 87.6%. This is attributed to the contributions of the API Retriever and fine-tuning. It is worth noting that ChatGPT, utilizing ground truth APIs, already achieves satisfactory results, while ToolLLaMA performs slightly worse due to limitations imposed by model size.

API Retriever For the base InternLM without fine-tuning, the introduction of the API Retriever results in decreased performance, dropping from 76.67% to 53.3%, which can be attributed to several factors. Firstly, the lower recall precision introduces cumulative errors in the decision-making process. Secondly, in the ToolBench dataset, there are numerous APIs in ground truth API set that can be replaced by semantically similar but functionally different APIs. While API Retriever may select many such APIs, the lack of Demo Selector results in a reduced score due to a limited understanding of the overall API call flow.

LLM Finetuner 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.

C.4 Cases of Real-World Scenario

To make the concept of our framework clearer, we present some real test examples, including the input and output sequences, in Table 6.

Table 6: Real examples of our framework.

User Query: Implementing surveillance on target person A.

LLM Output:
question: Implementing surveillance on target person A.
question_01: I need to know who is person A.
action_01: Lib_manage.Create('person A')
observation_01: ["Lib_manage('person A')"]
question_02: I need to implement surveillance on person A.
action_02: Surveillance.Task_manage(Lib_manage('person A'))
observation_02: ["New surveillance has been added."]
question_03: I need to surveil person A.
action_03: Alert_center.Surveillance_alert()
observation_03:["No warnings"]
summary: ["New surveillance has been added on person A, no warnings!"]

Agent Output:
New surveillance has been added on person A, no warnings!

User Query: I want to analyze suspect A's associates. Tell me about his relationships.

LLM Output:
question: I want to analyze suspect A's associates. Tell me about his relationships.
question_01: I need to know who is A.
action_01: Lib_manage.Search('A')
observation_01: ["Lib_manage('A')"]
question_02: I need to know the profile of A.
action_02: Holographic_profile.Person_profile_search(Lib_manage('A'))
observation_02: ["Holographic_profile.Person_profile('A')"]
question_03: I need to know the relationship of A.
action_03: Holographic_profile.Person_profile('A').Relationship_graph()
observation_03:["Person B is his ...", "Person C used to ...."]
summary: ["Person B is his ...", "Person C used to ...."]

Agent Output:
Person B is his .... Person C used to ....