Enhancing Large Language Model's Capabilities in Open Domains via Autonomous Tool Integration from GitHub

Anonymous ACL submission

Abstract

Large Language Models (LLMs) excel in traditional natural language processing tasks but struggle with problems that require complex domain-specific calculations or simulations. While equipping LLMs with external tools to build LLM-based agents can enhance their capabilities, existing approaches lack the flexibility to address diverse and ever-evolving user queries in open domains. Currently, there is also no existing dataset that evaluates LLMs on open-domain knowledge that requires tools to solve. Notably, the largest open-domain platform is GitHub. To this end, we introduce **OpenAct** based on human expert consult and repositories in GitHub. It comprises 339 questions spanning 7 diverse domains that need to be solved with domain-specific methods. In our experiments, even state-of-the-art LLMs and LLM-based agents demonstrate shallow success rates on OpenAct, underscoring the need for a novel approach. Based on the characteristics of this task, we present **OpenAgent**, a novel LLM-based Agent system that can tackle evolving queries in open domains through autonomously integrating specialized tools from GitHub. OpenAgent employs 1) a hierarchical framework where specialized agents handle specific tasks and can assign tasks to inferior agents, 2) a bi-level experience learning mechanism to learn from both humans' and its own experiences to tackle tool flaws. Experiments demonstrate OpenAgent's superior effectiveness and efficiency that significantly outperforms current methods.

1 Introduction

Large Language Models (LLMs) have demonstrated exceptional capabilities through diverse traditional natural language processing (NLP) tasks (OpenAI, 2022, 2023; Team et al., 2023; Anthropic, 2024). However, LLMs still struggle with specialized tasks that require calculation, simulation, data augmentation, etc (Qin et al., 2024). To tackle it, researchers equip LLMs with external tools (e.g., search engines (Nakano et al., 2021; Qin et al., 2023a), code executors (Qian et al., 2023b; Cai et al., 2023b), scientific simulators (Liu et al., 2022; Bran et al., 2023b)) to function as agents that are capable of solving complex tasks and extend the capability boundary of LLMs beyond traditional NLP tasks. Existing LLM-based agents (AutoGPT, 2023; Wu et al., 2023; XAgent, 2023; Schick et al., 2023; Parisi et al., 2022; Patil et al., 2023; Hong et al., 2024) have access to a pre-defined toolset and can combine its cognitive abilities with the specialized functionalities of these tools.

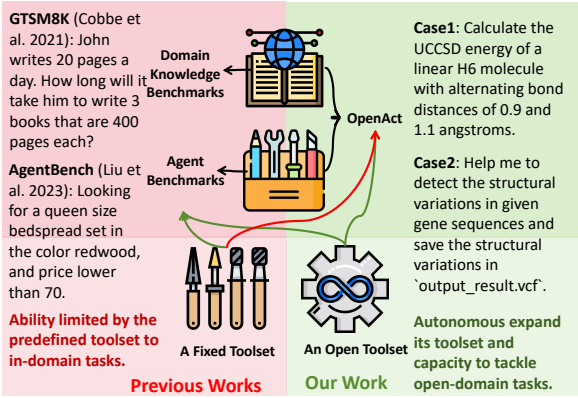

Figure 1: The comparison between our work and previous studies from both dataset and model perspectives. ↗ indicates that a model is capable of solving tasks within a benchmark, whereas ↗ indicates the opposite.

However, the effectiveness of current LLM Agents is constrained by the pre-defined toolset they rely on. This design paradigm restricts them to addressing in-domain problems while lacking the generalization capability to handle diverse open-domain questions, which is a critical limitation given that real-world user queries often require domain-specialized tools. This contradiction raises our core research problem: **Are LLM agents able to autonomously search for and adapt new tools for open-domain tasks?**

Addressing this challenge requires reevaluating existing evaluation paradigms. Current benchmarks are constructed based on pre-defined toolsets rather than real-world demands, creating an artificial performance ceiling that fails to assess models' ability to handle tasks requiring external specialized tools. Real-world tasks often necessitate specialized tools and domain-specific knowledge that extend beyond the inherent capabilities of pre-trained language models and a fixed toolset. Such tasks, like gene mutation detection, quantum chemistry analysis, and financial modeling, are typically executed by domain experts utilizing sophisticated professional tools and software.

In this context, GitHub emerges as it is the largest platform that contains implementations of algorithms and methodologies in open domains that are employed by experts in their respective fields. If LLM-based agents could effectively search for, deploy, and utilize relevant repositories from GitHub, they could independently extend their toolset. This capability would enable LLM agents to dynamically adapt and grow their abilities, significantly enhancing their versatility and effectiveness in addressing complex, real-world applications.

To this end, we introduce OpenAct. Its construction began by identifying key issues and methodologies across multiple domains. We then collected relevant tools on GitHub and carefully designed a series of tasks that reflect domain needs. OpenAct is the first dataset designed to evaluate LLMs on fulfilling open-domain real-world tasks. It comprises 339 queries across 7 diverse domains, including finance, chemistry, bioinformatics, computer vision, etc. Selected tasks in traditional benchmarks and ours are listed in Figure 1.

Experiments show that both vanilla LLMs and general-purpose LLM agents perform poorly on OpenAct. The key challenges are: (1) Lack of Quality Assurance: GitHub repositories often contain flaws, bugs, or incomplete/misleading documentation, (2) Alignment Gap: GitHub tools require adjustments to fit user needs, (3) Workflow Complexity: The process involves many diverse tasks, making it hard for LLMs to stay effective. Notably, (2) contributes to (3).

To solve these challenges, we introduce **OpenAgent**, a novel LLM-based agent system that autonomously extends tools from GitHub. It possesses two key features: (1) a hierarchical structure that dynamically decomposes the whole process into distinct subtasks (e.g. setting up environments, reading Issues/PRs), and (2) a bi-level experience learning mechanism to accumulate both in-task and cross-task experiences. Generally, OpenAgent starts with searching suitable repositories, then setting up the necessary environment, and utilizing the repository to fulfill user queries.

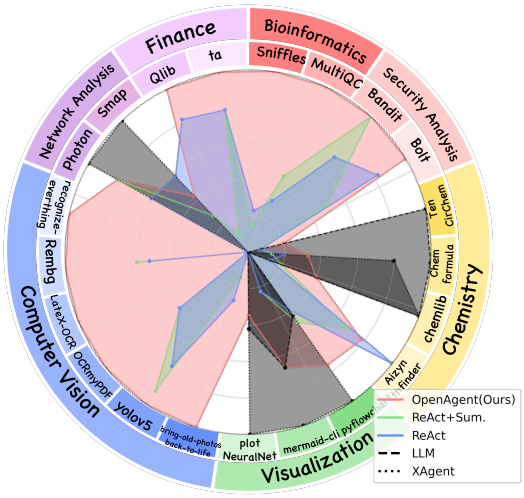

Figure 2: Illustration of GPT-4-based OpenAgent performs against baselines on 339 queries from 7 domains in OpenAct.

We conduct extensive experiments on OpenAct to demonstrate the effectiveness of OpenAgent compared to state-of-the-art LLMs and LLM Agents. Ablation studies prove the necessity of both the above two key features.

In summary, our contributions are threefold:

- We introduce OpenAct, a comprehensive dataset comprising 339 queries across 7 diverse domains, which is specifically designed to evaluate the capabilities of LLMs' open-domain capability in real-world scenarios.
- We propose OpenAgent, a novel LLM Agent system that autonomously extends its toolset by integrating repositories from GitHub. It employs a hierarchical structure and possesses a bi-level experience learning mechanism.
- Experiments demonstrate the effectiveness of OpenAgent and its two key features.

2 Related Work

LLM-based Agents. Large Language Models (LLMs) (OpenAI, 2022, 2023; Touvron et al.,

Table 1: Comparison of Benchmarks for Evaluating LLMs on Domain Knowledge and Tool Utilization. The "Domain Num." column indicates the number of domains evaluated by each benchmark, with "-" denoting benchmarks that do not assess domain knowledge. "Open End" denotes the presence of an open-ended environment for exploration within the benchmark. "Repository-Level" specifies whether the tasks in the benchmark are scoped at the repository level, with the number in the bracket denoting the number of repositories relevant to the benchmark.

Benchmark	Domain Num.	Task Source	Task Types	Code Use	Tool Use	Open End	Repository-Level
Minedojo (Fan et al., 2022)	-	Internet	Action	✓	✓	✓	✗
OSWorld (Xie et al., 2024)	-	Internet	Action	✗	✓	✓	✗
ToolBench (Qin et al., 2023b)	-	Tool	QA	✗	✓	✗	✗
MetaTool (Huang et al., 2024b)	-	Tool	QA	✗	✓	✗	✗
AgentBench (Liu et al., 2023)	-	Tool	QA	✓	✓	✗	✗
GTSM8K (Cobbe et al., 2021)	1	Domain	QA	✗	✓	✗	✗
ScienceQA (Lu et al., 2022)	3	Domain	QA	✓	✗	✗	✗
SciEval (Sun et al., 2023)	3	Domain	QA	✗	✗	✗	✗
SciBench (Wang et al., 2024)	3	Domain	QA	✓	✗	✗	✗
SWE-Bench (Jimenez et al., 2024)	1	GitHub	Coding	✓	✓	✗	✓(12)
ML-Bench (Tang et al., 2024)	1	GitHub	Coding	✗	✓	✗	✓(14)
SUPER (Bogin et al., 2024)	-	GitHub	QA	✓	✓	✗	✓(45)
OpenAct (Ours)	7	Domain and Github	QA and Coding	✓	✓	✓	✓(21)

2023a,b) have demonstrated remarkable proficiency across traditional natural language processing (NLP) tasks. LLM-based agents (AutoGPT, 2023; Wu et al., 2023; Li et al., 2023; XAgent, 2023) are LLMs equipped with external tools that can accomplish tasks requiring complex calculations or real-time actions (Yao et al., 2022a; Cheng et al., 2024; Park et al., 2023; Ye et al., 2023; Ma et al., 2024; Cai et al., 2023a; Wang et al.; Bogin et al., 2024; Kumar et al., 2023; Liu et al.; Bran et al., 2023a; Huang et al., 2024a; Qi et al., 2024; Kraus et al., 2023; Koldunov and Jung, 2024; Thulke et al., 2024; Vaghefi et al., 2023). However, existing research typically supports a limited set of tools, which cannot meet the diverse demands of humans. Recently, there has been a focus on tool creation (Cai et al., 2023b; Qian et al., 2023b; Wang et al., 2023; Qian et al., 2023a) for agents to dynamically create tools, which are typically file-level code scripts. The functionalities of these created tools remain simple and limited, insufficient to meet complex real-world user queries.

Benchmarking LLMs on Domain Knowledge, Tool Use and Open-Ended Tasks. Different benchmarks evaluate LLMs across diverse domains and capabilities. Domain knowledge benchmarks initially focused on mathematics (Cobbe et al., 2021; Hendrycks et al., 2021). Subsequent works(Lu et al., 2022; Sun et al., 2023) broadened the scope to encompass three domains: mathematics, physics, and chemistry. (Wang et al., 2024) further advanced this approach by incorporating code interpreter functionality while continuing to focus on these three domains. These benchmarks are typically derived from established knowledge sources such as textbooks and curated problem repositories, which do not fully capture real-world complexities or cutting-edge questions in rapidly evolving fields.

In parallel, tool use datasets (Qin et al., 2023b; Huang et al., 2024b; Liu et al., 2023) are designed based on the functionalities of pre-defined tools and APIs. Recent works have begun to bridge the gap between domain knowledge and practical application by focusing on coding tasks derived from real-world GitHub repositories. However, their scope remains limited to specific domains (Jimenez et al., 2024; Tang et al., 2024; Bogin et al., 2024).

Lastly, while open-ended exploration is crucial for real-world tasks, existing research has primarily studied it in action-oriented environments (Fan et al., 2022; Wang et al., 2023; Xie et al., 2024).

In conclusion, existing benchmarks remain limited in their scope, domains, tool use, or coding tasks in isolation. They frequently lack the combination of multimodality, code use, tool utilization, and open-ended exploration that characterizes many real-world problem-solving scenarios. Table 1 lists the main differences between our benchmark and previous works.

3 Dataset: OpenAct

3.1 Dataset Construction

We present OpenAct, a high-quality benchmark spanning 7 professional domains that bridge open-domain knowledge with executable implementation resources. As is shown in Figure 3, the dataset construction process involves the following stages:

Domain-Specific Problem Curation. We collaborated with domain experts to identify frontier challenges within their respective fields that are

Figure 3: The construction pipeline of OpenAct.

amenable to computational solutions. This process yielded 7-10 candidate problems per domain through iterative refinement with expert feedback.

Repository Selection and Filtering. We conducted systematic searches on GitHub to identify repositories implementing solutions for the curated problems. From an initial pool of 10 repositories per domain, we applied rigorous filtering criteria: (1) Removal of repositories with duplicate functionality or implementation approaches; (2) Verification of active maintenance status within the past 6 months; (3) Assessment of documentation quality. This yielded 21 repositories across seven domains.

Query Generation and Validation. For each repository, we employed GPT-4 to generate 30 candidate queries of varying complexity levels. Through manual validation we retained 5-10 executable queries per repository, ensuring: (1) Solvability using the target repository; (2) Coverage of different functionality aspects; (3) Absence of ambiguous phrasing. This process resulted in 113 validated base queries.

Problem Enhancement. To investigate information hinting strategies, we designed three prompt conditions for each base query: (1) *Explicit Hint*: Direct repository specification with GitHub URL; (2) *Implicit Hint*: Domain/keyword-based hints without repository identification; (3) textitNo Hint: Base query without supplemental information.

This multi-condition approach yields 339 instruction-answer pairs with expert-validated ground truth solutions. Table 2 summarizes the dataset statistics across domains. Appendix A contains more detailed statistics of OpenAct.

Table 2: Statistics of OpenAct.

Domain	Num. of Repo.	Num. of Query
Finance	2	45
Chemistry	4	66
Bioinformatics	2	30
Computer Vision	6	90
Network Analysis	2	30
Security Analysis	2	30
Visualization	3	48
Total	21	339

3.2 Data Categorization

We categorized the collected repositories based on the difficulty of the Setup and Apply phases.

For the **Setup difficulty**, we divided the collected repositories into 3 classes: (1)*Setup-Easy*: The README provides a detailed and correct setup tutorial, with which the environment can be set up fluently. (2)*Setup-Medium*: The README misses some details or contains slight flaws, which requires the agent to solve based on error reports. (3)*Setup-Hard*: The README provides an incorrect tutorial because of human error or insufficient maintenance, which needs agents to find relevant Issues/PRs to solve.

For the **Apply difficulty**, we also divided the repositories into 3 classes: (1)*Apply-Easy*: Simply requires running some commands given by the README. (2)*Apply-Medium*: Requires writing configuration files or downloading extra resources, like data and trained models. (3)*Apply-Hard*: Requires modifying the source code of the repositories or need to refer to relevant Issues/PRs for help.

3.3 Operation Environment

To ensure experiment reproducibility and minimize dependency on local environments, we designed a comprehensive interface that allows LLMs to interact seamlessly with a Docker container. In our work, all interactions with by the LLMs are executed within a controlled Docker environment.

3.4 Evaluation Metrics

We designed 2 evaluation metrics for tasks in OpenAct: Completeness and Pass Rate.

Pass Rate The Pass Rate is defined as the proportion of queries that successfully meet the predefined criteria relative to the total number of queries. This evaluation is conducted exclusively based on the comparison of the final answer with the expert-generated "golden answers" by GPT-4. A query is deemed to pass if there is a concordance between these two answers.

Completeness To precisely evaluate the performance of OpenAgent and its baselines, we fur-

ther designed a metric to evaluate the whole execution process with a GPT-4-based evaluation agent, scoring from 0 to 10. The evaluation covers three phases: Search, Setup, and Apply. GPT-4 assigns scores of $[0, 3]$ for Search, Setup, and Apply, and $[0, 1]$ for the final answer. These scores are subsequently aggregated and normalized to a 10-point scale to derive the overall completeness score.

We sampled 120 queries and results for human annotation, achieving an 87.5% absolute match with GPT-4 evaluations, indicating the high reliability of our metrics. The details of this check are depicted in Appendix B.

4 OpenAgent

Our preliminary experimental results show that both vanilla LLMs and general-purpose LLM agents perform poorly on OpenAct. Even when we provide the LLM with relevant GitHub interfaces and a well-constructed environment to create a custom LLM agent, the model's performance remains inferior We attribute it to several challenges in employing GitHub repositories to fulfill a task: (1) **Lack of Quality Assurance**: GitHub repositories often lack standardization and may contain flaws or bugs, and their documentation may also be incomplete, misleading, or containing errors. (2) **Alignment Gap between Tools and Queries**: Tools on GitHub need adjustments to suit the users' needs. (3) **Workflow Complexity**: The whole workflow involves dozens of different tasks. The significant differences between these tasks can easily distract the LLMs from completing the whole process effectively.

To address this, we propose OpenAgent, with 2 novel features: (1) Hierarchical Agent System, where models delegate subtasks to reduce workflow burden (Section 4.1), (2) Bi-Level Experience Learning (Section 4.2), which learns from Issues/PRs for in-task knowledge and accumulates cross-task experience. The first layer of the Hierarchical Agent System consists of three main components and we introduce them in Section 4.3.

4.1 Hierarchical Agent System

As shown in the figure, the Hierarchical Agent System breaks down complex tasks into smaller sub-tasks, where each agent receives a query from a higher-level entity and responds by either taking direct action or delegating tasks to sub-agents. These interactions fall into two categories: (1) Action Calls: Direct interactions with the environment, such as executing commands, checking files, or submitting results. (2) Agent Calls: Assigning sub-tasks to specialized agents, such as setting up environments or modifying files.

Each agent processes its query based on previous actions and observations. If it delegates to a sub-agent, the sub-agent continues the process with a refined query. If the agent acts, it either completes the task (reporting the result back) or interacts with the environment and records feedback.

This recursive process continues until all tasks are broken down into simple, executable actions. By structuring tasks hierarchically, the system efficiently manages complexity, allowing each agent to focus on its specific role while collectively achieving intelligent behavior.

4.2 Bi-Level Experience Learning

We developed and implemented an experience learning feature for OpenAgent, encompassing both in-task and cross-task learning paradigms.

Due to the non-standardization of GitHub repositories, some lack perfect READMEs and necessary setup information. Additionally, flaws in the source code pose challenges. In such cases, learning from human experiences becomes an efficient approach.

Building upon the hierarchical framework, we introduce a specialized agent, the Issue/PR Agent to handle the experience learning process. This agent is called when a higher-level agent encounters a problem that might benefit from past experiences or community solutions. It is responsible for searching, evaluating, and returning relevant information from GitHub Issues and Pull Requests.

Apart from in-task knowledge learned from com-

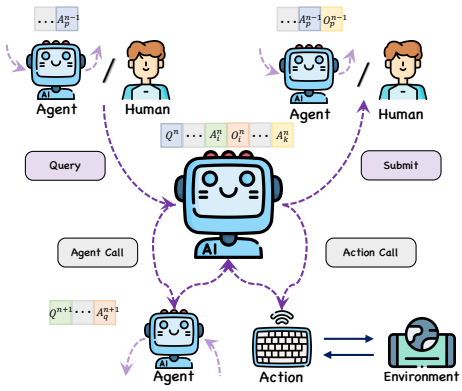

Figure 4: Illustration of the Hierarchical Agent System, where blocks mean memory list and same background color denote same information.

Methods	Finance	Chemistry	Bioinformatics	Computer Vision	Network Analysis	Security Analysis	Visualization	Avg.
GPT-3.5-Turbo Based								
Vanilla	0	**36.4**	0	0	0	0	**31.3**	11.5
ReAct	2.2	3.0	3.3	6.7	0	0	0	2.4
ReAct + Sum.	0	0	0	0	0	0	0	0
OpenAgent (Ours)	**8.9**	24.2	**23.3**	**8.9**	**10.0**	**33.3**	20.1	**17.1**
GPT-4 Based								
Vanilla	0	**68.2**	0	0	0	0	43.8	19.5
XAgent	0	40.9	0	0	**40.0**	0	**81.3**	23.0
ReAct	51.1	19.7	17.8	22.2	10.4	30.0	23.3	24.6
ReAct + Sum.	31.1	19.7	26.7	22.9	14.8	33.3	26.7	24.4
OpenAgent (Ours)	**68.9**	34.9	**86.7**	**45.6**	16.7	**43.3**	35.4	**47.3**

Table 3: Pass Rates (%) of different methods across various domains in the OpenAct dataset. Results are shown for both GPT-3.5-Turbo and GPT-4-based implementations. "All" represents the overall pass rate across all domains.

munity experiences and solutions, OpenAgent can also learn from its own past experiences to improve decision-making over time. It updates its knowledge by summarizing past actions and outcomes, refining its approach for future tasks.

After completing a task, the agent stores its execution environment in a Docker image, allowing for easy reuse when similar queries arise. To enhance retrieval, it abstracts the repository's functionality and summarizes key experiences, ensuring efficient adaptation to new challenges.

4.3 Main Phases

Repository Search. In the search phase, the agent identifies suitable repositories to fulfill the user query. It first checks previously stored repositories to determine if any match the query. If a suitable repository is found, its environment is loaded directly, bypassing the setup phase. If no stored repository is available, the agent searches GitHub for relevant options. If the user specifies a repository, the agent retrieves it by name; otherwise, it searches by topic to find relevant repositories. Once candidate repositories are gathered, the agent evaluates their suitability by analyzing their README files and determining whether they can effectively address the user's request.

Environment Setup. Once a suitable repository is identified, the agent sets up its execution environment. This begins with cloning the repository and installing dependencies as outlined in its README file. However, since many repositories lack standardized documentation or contain flaws, the agent may need to search GitHub Issues and Pull Requests to resolve problems. If necessary, it modifies the repository's source files to fix bugs. To ensure security and isolation, all operations are conducted within a Docker environment.

Tool Application. After configuring the environment, the agent proceeds to apply the repository to solve the user query. If the repository is well-structured and provides a clear entry point, such as a command-line interface, the agent can use it directly. However, for non-standardized repositories with limited documentation, the agent relies on human experiences extracted from Issues and PRs. If the execution produces extensive output, the agent writes a Python script to extract key information efficiently. Elaborated introduction to these phases are in Appendix A.

Note that although this process follows a hierarchical structure, the agent dynamically decides which phase, subtask, or action to execute based on the specific query. This flexibility ensures adaptability to diverse and complex tasks.

5 Experiment

5.1 Experiment Settings

Baselines. To validate the effectiveness of our OpenAgent, we design the following baselines: (1) LLM: Vanilla LLMs without external tools (2) ReAct (Yao et al., 2022b): ReAct is a widely-used LLM-based agent task-solving technique (AutoGPT, 2023; Wu et al., 2023). In our settings, ReAct is equipped with the same actions as our OpenAgent to extend tools from GitHub for fair comparison. (3) ReAct+Summary: Due to the complexity of the tool extension, the whole process tends to involve lengthy context, surpassing the context window of LLMs. Hence, we design this ReAct variant which will summarize the context when the length of the context reaches the threshold. (4) XAgent (XAgent, 2023): XAgent is a powerful general-purposed LLM-based agent, which

Method	Pass Rate
OpenAgent w/ PRs&Issues	47.3
OpenAgent w/o PRs&Issues	40.3

Table 4: Results of ablating in-task experience learning.

Method	w/o SelfExp	w/ SelfExp
GPT-3.5	17.6	58.8
GPT-4	47.0	82.3

Table 5: Results of employing cross-task experience learning.

is equipped with numerous external tools and can reason, plan, code, and reflect.

Implementation Details We implement OpenAgent and baseline methods except XAgent based on the `gpt-4-0125-preview` and `gpt-3.5-turbo-16k` respectively with a 0.7 temperature, under a 0-shot setting. There is no GPT-3.5-based XAgent because its behindhand reasoning and planning ability can't support XAgent's complex workflows. **Specially**, the following main experiments reflect the success rate on the first encounter with the problem, incorporating PR/Issue but without using experience summary (cross-task experience learning) to enhance the strategy.

5.2 Overall Results

Table 3 reports the Pass Rates of each method. We get several observations: (1) While Vanilla LLMs and XAgent demonstrate good performance in familiar domains like Chemistry and Visualization, it is impossible for them to fulfill questions in unacquainted domains like Bioinformatics, Finance, etc. (2) ReAct achieves a lower Pass Rate than the agency structure in both settings, which demonstrates that simply adapting the ReAct framework cannot achieve good results. (3) ReAct+Summary achieves lower performance than ReAct because the summarization will lose critical information. Thus, it is infeasible to avoid the over-length problem by simply summarizing the long context. (4) All GPT-4-based methods outperform their GPT-3.5 counterparts significantly, showing that OpenAct is a challenging dataset needing powerful LLMs to achieve. (5) OpenAgent with hierarchical architecture significantly outperforms all baselines and incurs the least computation cost in both settings, demonstrating the effectiveness and efficiency of the Hierarchical Agent System.

5.3 Abalation Study

To evaluate the effectiveness of the bi-level experience learning mechanism, we conduct the ablation study. For **in-task experience learning**, we remove the PRs/Issues actions to re-run the main experiments. Experimental results are shown in Table 4 and we observe that without PRs/Issues, the pass rate decreases to 47.3%. It not only verifies the non-standardization problem of GitHub repositories but also proves that learning from PRs/Issues can overcome this challenge.

For **cross-task experience learning**, We select 2 repositories: `Qlib` and `AiZynthFinder`, which both belong to the Hard Apply category. We run their 51 queries and utilize the GPT-4-based OpenAgent to store the repositories with summarized practice experience. We then re-run these queries but OpenAgent would retrieve the stored repositories and utilize the summarized experience to accomplish the queries. As shown in Table 5, leveraging the experience summarized by GPT-4-based OpenAgent, GPT-3.5-based OpenAgent can even achieve a higher pass rate than GPT-4-based OpenAgent without summarized experience. Simultaneously, GPT-4-based OpenAgent can achieve a higher Pass Rate even though it leverages the summarized experience by itself.

The above studies prove the effectiveness of bi-level experience learning.

5.4 Impact of Different Phases

We delve deeper into each phase from a thorough understanding of OpenAct and OpenAgent.

Search. As introduced in Section 3.1, we design three types of prompts to denote the target repositories. We calculate the Seach Success Rate of each type of prompt. If OpenAgent can get the correct repositories of a query, it is denoted as search success. Then, we calculate the proportion of the search success queries over all queries of each type of prompt. The experimental results are listed in Table 6. We can observe that the Explicit Repo Prompt achieves the highest Search Success Rate (near 100%) as the prompt has specified repositories. Implicit Repo Prompt achieves 66.0% Search Success Rate, showing that OpenAgent can infer the relevant GitHub Topics based on the domains or careers. Finally, if no repository prompt is provided, the search success rate decreases significantly. It demonstrates that OpenAgent falls short

of inferring GitHub Topics based on the query only. It needs further research in the future to improve the performance in this situation.

Table 6: Analysis for the search difficulty.

Prompt	Search Success Rate
Explicit Repo Prompt	96.0
Implicit Repo Prompt	66.0
No Repo Prompt	32.0

Setup & Apply. Table 7 shows the Pass Rates for the repositories categorized based on Setup and Apply difficulties, as described in Section 3.2.

For Setup difficulty, both Medium and Hard repositories achieve similar Pass Rates. We attribute this to OpenAgent's human experience learning capability that helps overcome imperfect READMEs.

For Apply difficulty, the Pass Rate for Hard decreases by over 12% compared to Easy and Medium. This demonstrates that while OpenAgent can effectively handle repositories with easy and medium Apply difficulty, it requires further study to conquer those with hard Apply difficulty.

Table 7: Analysis for the setup & apply difficulty.

Setup/Apply Difficulty	Easy	Medium	Hard	Total
Easy	72.3	69.0	56.2	64.4
Medium	60.7	70.0	41.5	57.7
Hard	50.0	67.0	51.5	57.4
Total	64.1	68.7	51.4	60.7

5.5 Error Analysis

Although our method can autonomously extend tools from GitHub, we still observe some failures.

Repository Select Failure A key challenge for OpenAgent is selecting the appropriate repository from GitHub to address user queries. We noticed instances where OpenAgent selected repositories that were not capable of resolving the given queries. This issue was particularly prevalent when the user query did not specify a particular repository. Then the agent's decision-making process relies heavily on the README files of repositories. However, these files sometimes lack clear and explicit descriptions of the repository's functionality, or overclaim its functionality, and lead to misjudgments by the agent. For example, in the Finance scenario, OpenAgent erroneously selected the `vnpy` repository, which is suited for quantitative trading but not for research applications like setting specific models (e.g., LSTM).

Environment Configuration Failure In cases like `Bringing-Old-Photos-Back-to-Life`, we observed failures in the environment setup. While an official Dockerfile was present, it was outdated and non-functional without enough maintenance. The correct Dockerfile was located within a Pull Request, which the agent should ideally access to find the accurate setup instructions. However, OpenAgent sometimes opted to modify the existing, incorrect Dockerfile rather than seeking the correct version in PRs. Due to unresolved bugs in the Dockerfile, the agent was unable to correctly set up the environment, leading to failure.

Execution Configuration Failure The repository `Qlib` presented unique challenges, as it requires writing a specific configuration file for execution. This file encompasses a range of parameters including dataset settings, model hyperparameters, and backtesting parameters. Incorrect settings in any of these parameters can lead to results that do not meet the user query's requirements. In practice, we observed that OpenAgent may incorrectly set the time range for data or specified erroneous file paths, resulting in execution failure.

These observed failures primarily stem from the quality and complexity of the repositories. These insights underscore the need for further improvements in OpenAgent's robustness and decision-making algorithms to enhance its success rate in diverse and complex scenarios.

We also conduct Case Study in Appendix D to further show the detailed process of OpenAgent.

6 Conclusion

In this paper, we introduced OpenAct, a comprehensive dataset designed to evaluate the capabilities of LLMs in open-domain, real-world scenarios. Our experiments highlighted the limitations of existing LLM-based agents and demonstrated the effectiveness of our proposed OpenAgent system. OpenAgent's hierarchical framework and bi-level experience learning mechanism significantly enhance its capabilities, allowing them to tackle complex tasks across diverse domains. Our work paves the way for more robust and flexible LLM-based agents, capable of evolving alongside rapidly changing technological landscapes.

7 Limitation

Our study has explored the tool extension capability of LLM-based agents, yet both researchers and practitioners must be mindful of certain limitations and risks when using the approach to develop new techniques or applications.

Firstly, our method relies on the utilization of Pull Requests (PRs) and Issues from GitHub as primary sources of human experience. However, it is important to recognize that similar functionalities may not be universally available across other repository hosting platforms. Consequently, the generalizability of our findings is confined to the GitHub ecosystem, posing a limitation to the applicability of our approach beyond this specific context.

Secondly, the dynamic nature of GitHub repositories, characterized by frequent updates, bug fixes, and the evolution of repository functionalities, introduces a layer of volatility. This fluidity can significantly impact the reproducibility of our experimental results over time, as the state of the repositories at the time of study may not reflect their future states.

Thirdly, our method necessitates the use of Docker for the execution of repository content. This dependency on Docker implicates substantial consumption of server resources, including CPU, memory, and storage. Such resource-intensive requirements may pose practical limitations on the scalability and feasibility of deploying OpenAgent, particularly in environments with constrained computational resources.

References

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf. Accessed: 2025-01-30.

AutoGPT. 2023. Autogpt.

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. 2024. Super: Evaluating agents on setting up and executing tasks from research repositories.

A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller. 2023a. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2023b. Chemcrow: Augmenting large-language models with chemistry tools.

T. Cai, X. Wang, T. Ma, X. Chen, and D. Zhou. 2023a. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023b. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

K. Cheng, Q. Sun, Y. Chu, F. Xu, L. YanTao, J. Zhang, and Z. Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework.

K. Huang, Y. Qu, H. Cousins, W. A. Johnson, D. Yin, M. Shah, D. Zhou, R. Altman, M. Wang, and L. Cong. 2024a. Crispr-gpt: An llm agent for automated design of gene-editing experiments. *arXiv preprint arXiv:2404.18021*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024b. Metatool benchmark for large language models: Deciding whether to use tools and which to use.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

N. Koldunov and T. Jung. 2024. Local climate services for all, courtesy of large language models. *Communications Earth & Environment*, 5(1):13.

M. Kraus, J. Bingler, M. Leippold, T. Schimanski, C. C. Senni, D. Stammbach, S. Vaghefi, and N. Webersinke. 2023. Enhancing large language models with climate resources. Technical report, Swiss Finance Institute.

V. V. Kumar, L. Gleyzer, A. Kahana, K. Shukla, and G. E. Karniadakis. 2023. Mycrunchgpt: A chatgpt assisted framework for scientific machine learning. *Journal of Machine Learning for Modeling and Computing*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large scale language model society.

R. Liu, J. Wei, S. S. Gu, T. Wu, S. Vosoughi, C. Cui, D. Zhou, and A. M. Dai. Mind's eye: Grounded language model reasoning through simulation. In *The Eleventh International Conference on Learning Representations*.

Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou, and Andrew M. Dai. 2022. Mind's eye: Grounded language model reasoning through simulation.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents.

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering.

P. Ma, T. Wang, M. Guo, Z. Sun, J. B. Tenenbaum, D. Rus, C. Gan, and W. Matusik. 2024. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *International Conference on Machine Learning*. PMLR.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv preprint*, abs/2112.09332.

OpenAI. 2022. OpenAI: Introducing ChatGPT.

OpenAI. 2023. Gpt-4 technical report.

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

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.

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*.

J. Qi, Z. Jia, M. Liu, W. Zhan, J. Zhang, X. Wen, J. Gan, J. Chen, Q. Liu, M. D. Ma, B. Li, H. Wang, A. Kulkarni, M. Chen, D. Zhou, L. Li, W. Wang, and L. Huang. 2024. Metascientist: A human-ai synergistic framework for automated mechanical metamaterial design.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023a. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023b. Creator: Disentangling abstract and concrete reasonings of large language models through tool creation. *arXiv preprint arXiv:2305.14318*.

Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, Y. Huang, C. Xiao, C. Han, Y. R. Fung, Y. Su, H. Wang, C. Qian, R. Tian, K. Zhu, S. Liang, X. Shen, B. Xu, Z. Zhang, Y. Ye, B. Li, Z. Tang, J. Yi, Y. Zhu, Z. Dai, L. Yan, X. Cong, Y. Lu, W. Zhao, Y. Huang, J. Yan, X. Han, X. Sun, D. Li, J. Phang, C. Yang, T. Wu, H. Ji, Z. Liu, and M. Sun. 2024. Tool learning with foundation models.

Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, et al. 2023a. Webcpm: Interactive web search for chinese long-form question answering. *arXiv preprint arXiv:2305.06849*.

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*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv preprint*, abs/2302.04761.

Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhennan Shen, Baocai Chen, Lu Chen, and Kai Yu. 2023. Scieval: A multi-level large language model evaluation benchmark for scientific research.

Xiangru Tang, Yuliang Liu, Zefan Cai, Yanjun Shao, Junjie Lu, Yichi Zhang, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Liang Chen, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yin Fang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and

Mark Gerstein. 2024. Ml-bench: Evaluating large language models and agents for machine learning tasks on repository-level code.

G. Team, R. Anil, S. Borgeaud, Y. Wu, J. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

D. Thulke, Y. Gao, P. Pelser, R. Brune, R. Jalota, F. Fok, M. Ramos, I. Wyk, A. Nasir, H. Goldstein, et al. 2024. Climategpt: Towards ai synthesizing interdisciplinary research on climate change. *arXiv preprint arXiv:2401.09646*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

S. Vaghefi, Q. Wang, V. Muccione, J. Ni, M. Kraus, J. Bingler, T. Schimanski, C. C. Senni, N. Webersinke, C. Huggel, and M. Leippold. 2023. Chatclimate: Grounding conversational ai in climate science. *Swiss Finance Institute Research Paper No. 23-88*.

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

X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2024. Scibench: Evaluating college-level scientific problem-solving abilities of large language models.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

XAgent. 2023. Xagent: An autonomous agent for complex task solving.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments.

S. Yao, H. Chen, J. Yang, and K. Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc.

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

Y. Ye, X. Cong, S. Tian, J. Cao, H. Wang, Y. Qin, Y. Lu, H. Yu, H. Wang, Y. Lin, Z. Liu, and M. Sun. 2023. Proagent: From robotic process automation to agentic process automation.

Appendix

A Prompts

A.1 Main Agent

```
You are a professional programmer. Given a
query, your task is to search for a github
repository and use it to solve the query.

You should make sure the result of `apply`
function well completed the query. If it is
lack of required elements, you can call `apply`
 again if you think the result is close to what
 you want and you think this repository can be
used to solve your query. You can also call `
search_by_query` function to find another
repository if you think this repository is not
suitable for your query.
```

A.2 Search Agent

```
You are a professional programmer. Given a task
, you want to find a github repository to solve
 the task.
```

```
You are a professional programmer. Given a task
, you want to find a github repository to solve
 the task. Now, your colleagues have explored
some repositories. If you think any of the
repository(s) might can solve your task, call `
use_existing_repository` function to use it.
Otherwise, call `find_a_new_repository`
function to find another repository.

You will be given the query of the task and
name(s) and description(s) of existed
repositories.

Repository's name: {{name of repository 1}}
Description: {{description of repository 1}}

Repository's name: {{name of repository 2}}
Description: {{description of repository 2}}
......
```

A.3 Setup Agent

```
You are a professional programmer. Your task is
 to set up the environment of the repository
and prepare necessary data.

You will be provided with the readme file of
the repository. You can also use `
check_file_or_directory` function to check the
`/<==repo_name==>` directory whether there is a
 existed Dockerfile. If setting up the
environment is complex and there is an existing
 dockerfile, you can use `
set_container_with_existed_dockerfile` function
 to directly use that dockerfile. If there is
any problem with the dockerfile, you can try to
 use `read_pulls_to_solve_problem` function to
see the pulls of this repository to solve the
problem. However, `read_pulls_to_solve_problem`
 should not be used for reasons other than
troubleshooting issues with the Dockerfile. If
the existed dockerfile is built successfully,
you can call `submit` function directly with
property "work_directory" marked because the
required docker container has already been
built.

Usually the dockerfile is close to `/<==
repo_name==>`, so if you don't find it in one
or two try, it means there isn't a dockerfile
in this repository. You don't need to try more
times.

If there is no existing dockerfile, you should
analyze the readme file and derive the
necessary commands and execute them to set up
the environment of the repository and prepare
necessary data in a given container, whose base
 image is 'continuumio/miniconda3'. If error
happens due to inappropriate base image, you
can use `echo` to create a dockerfile yourself,
 with proper base image and necessary packages,
 and build it.

While operating, please note the following
points:
- The commands will be run in a docker
container. You don't need to use virtual
environments, use the base environment only.
Use pip or conda to install packages. In
special cases, you can use apt-get to install
necessary packages. If you use apt-get, do not
forget to use apt-get update and --fix-missing.
- Any command requiring execution in a specific
 directory should be reformulated as: `/bin/sh
-c "cd <specific directory> && <commands to be
executed in this directory>"`. Every command
must start with '/bin/sh -c " cd ' to locate a
specific directory.
- The repository have been clone to the root
directory at `/<==repo_name==>`.
- Follow the sequence of the commands, install
all necessary packages first.
- Never create or activate any conda
environment even if the readme requires or does
 so. You should install the packages in the
base environment.
- If you have problem with the version of
python, please reinstall python of the
appropriate version with `conda install python
=<version>`.
- If a function you called return you with a
file path, you should pass the file path to the
 next function you call if need.
- If there are different choices to do the same
 task and you failed using one of them, you can
 try another alternative.

Your commands should be the parameter of the `
execute_command` function. Each time you should
 send one or many commands. The `
execute_command` function will run the commands
 and return the output of the commands.

In this step, you should just set up the
environment and prepare the data. You don't
need to run other programs or train the model.
```

A.4 Apply Agent

```
You are a professional programmer. Your task is
 to utilize a github repository to solve a
given query. You will operate in a docker
container.

Note that it has been ensured that the
repository's environment has been set up and
all the data required by the readme has been
fully prepared, so you mustn't execute any
command to set up the environment or prepare
the data or check relevant files about the
environment or data anymore, unless the user
provide you with a link to download necessary
data. <==data_path==>

Also, all the dependencies have been installed
in the base environment, please don't switch to
 any other conda environment. If you find you
lack of any packages or tools while operating,
use pip, conda or apt-get to install it. If you
 use apt-get, do not forget to use `apt-get
update` and `--fix-missing`.

Your goal is to study the readme file
especially the command lines in it and call
appropriate functions to utilize the repository
 to solve the query. Do not execute any command
 to get result that you can't perceive yourself
, like starting a server.

Note that the default configuration of the
final executable file may not meet the demand
of the query. If there is any special demands
in the query, you should check the final
executable file to check whether it meets the
demand of the query. If not, you should make
proper modification(s).

If you run a command and find the result lack
of required element(s), which may because the
repository itself doesn't support relevant
function, you can check the issues to try to
solve the problem.

If you need to deal with files provided by the
user, you should firstly use `
upload_directory_to_container` to upload it
```

```
from local to the docker container. By default,
 the path claimed in the query is local path,
you need to upload it. If required message can
be retrieved from the output of execution of
the program, summarize it to natural language
and submit it. If any file is generated to
answer the query, you should use `
download_directory_from_container` to download
the file from the docker container to local
before you submit if necessary. You should also
 ensure required directories all exist before
running a program.

We only have CPU. If the repository doesn't ask
 for configuration of device, ignore it.

Readme:{{readme}}
```

A.5 Modify Agent

```
You are a professional programmer. Your task is
 to make modification(s) to code files to meet
the given requirement. You will be given the
query of modification, the content of a file
and the path to the file. If you think you can
meet the query through modifying this file, you
 can modify this file.

If the query contains path that contains
information for modification, transmit that
path at "query_file_path" in "
modify_entire_file". You don't need to check
the query file yourself, because you may
neglect important message by checking and
summarizing, just pass the query path and let "
modify_entire_file" function to decide.

Code relevant to the query may not always
reside in the currently provided file. In such
cases, you should analyze the `from...import
...` or `<module name>...` sections to suggest
potential target file paths.

If the target path in the current file is
relative path, you should decide the target
file based on the current files path.

If it starts from a module's name, which
suggests the file is a python package, the file
 is in `/opt/conda/lib/python3.11/site-packages
/<package name>` directory (python version
should be decided by using `which pip`). Don't
forget the suffix of the file.

You might need to locate the target file by
checking the content of the files recursively.
After the target file is located, you should
use proper functions to modify the code.
```

A.6 Judge Agent

```
You are a professional programmer. Your task is
 to judge how good a programmer use a github
repository to handle a query. You will be given
 query and the actions the programmer took to
handle the query. If the task includes input or
 output file, you will be given path to
programmer's output. Path to input is in the
query and path to the ground truth outcome will
 be given if there is ground truth. You can
check the content in these paths and use proper
 ways to judge the relevance of different files.
 If the files are readable you can directly
check them. If not, you can use the provided
functions to check the md5 hash value of the
files or compare the similarities of different
images. Note that you can only check directory
or file saved in local. If no input path,
output path, truth path is given, do not check
file or directory, just score based on the log.

// For ReAct & ReAct + Summary
The rule of scoring is as follows. The initial
score is 0. You will be given the log of user
calling functions to use the repository. For
correctly setting up the environment and
preparing the data, 2 point should be added for
 environment and 1 point should be added for
data. If no data is required, point for data
should be added.\nIn the given application
phase, 0~4 scores should be added based on the
performance. You should judge the performance
based on whether it follows the instruction in
the readme. If right actions(including commands
 and function calling) are taken and get a
result, you should add 4. If asked
configuration is not applied or wrong actions
are taken, minus 1 point for each fault based
on 4. If ground truth is provided, if the
result of the application is not correct, minus
 1 point.\nIn conclusion, the final score is
the sum of the scores of the setup (0~3) and
application phase (0~4).

// For GitAgent
The rule of scoring is as follows. The initial
score is 0. You will be given the log of user
calling functions to use the repository,
without the steps the environment is setup.\nIn
 the given application phase, 0~4 scores should
 be added based on the performance. You should
judge the performance based on whether it
follows the instruction in the readme. If right
 actions(including commands and function
calling) are taken and get a result, you should
 add 4. If asked configuration is not applied
or wrong actions are taken, minus 1 point for
each fault based on 4. If ground truth is
provided, if the result of the application is
not correct, minus 1 point.\nGenerally, if
valid output is given, the score should be 4.

Query:{{query}}

Action:{{action_log}}

Input path:{{input_path}}

Output path:{{output_path}}

Ground Truth path:{{truth_path}}
```

B GPT-4 Evaluation Alignment Checking

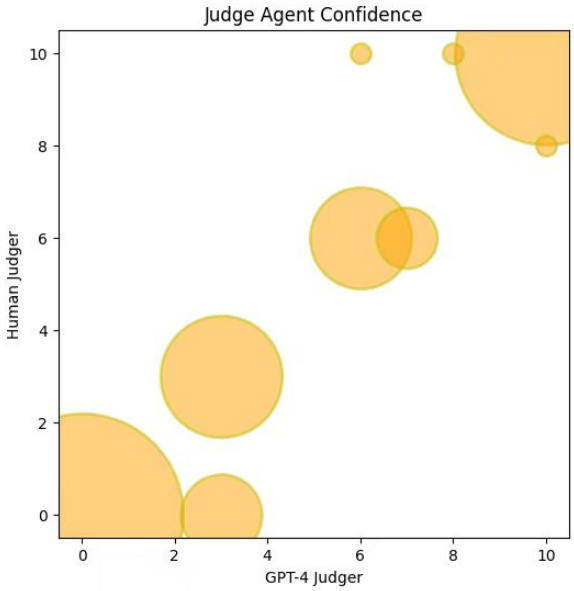

Figure 5: GPT-4 Evaluation Alignment Check

We randomly selected 120 questions from the OpenAct and conducted both manual scoring and machine scoring using our designed GPT-based Agent, then visualized the results in the above figure. The circular points represent the coordinates corresponding to human scores and machine scores, with the size of the circles indicating the number of questions with that particular score combination. As shown, there is a high consistency between machine scoring and human subjective assessment.

C Dataset Details

C.1 Overall Statistics

Please refer to Table 8 for the overall statistics regarding the repositories.

C.2 Repositories Categorized by Field

Please refer to Table 9 for the field of each Github repository.

C.3 Repositories Categorized by Difficulties

Please refer to Table 10 for the difficulty of each Github repository.

Table 8: GitHub Repositories

Author	Name	Address
danielgatis	rembg	https://github.com/danielgatis/rembg
ocrmypdf	OCRmyPDF	https://github.com/ocrmypdf/OCRmyPDF
cdfmlr	pyflowchart	https://github.com/cdfmlr/pyflowchart
HarisIqbal88	PlotNeuralNet	https://github.com/HarisIqbal88/PlotNeuralNet
lukas-blecher	LaTeX-OCR	https://github.com/lukas-blecher/LaTeX-OCR
s0md3v	Photon	https://github.com/s0md3v/Photon
s0md3v	Bolt	https://github.com/s0md3v/Bolt
s0md3v	Smap	https://github.com/s0md3v/Smap
MultiQC	MultiQC	https://github.com/MultiQC/MultiQC
xinyu1205	recognize-anything	https://github.com/xinyu1205/recognize-anything
bukosabino	ta	https://github.com/bukosabino/ta
molshape	ChemFormula	https://github.com/molshape/ChemFormula
tencent-quantum-lab	TenCirChem	https://github.com/tencent-quantum-lab/TenCirChem
harirakul	chemlib	https://github.com/harirakul/chemlib
ultralytics	yolov5	https://github.com/ultralytics/yolov5
mermaid-js	mermaid-cli	https://github.com/mermaid-js/mermaid-cli
microsoft	qlib	https://github.com/microsoft/qlib
fritzsedlazeck	Sniffles	https://github.com/fritzsedlazeck/Sniffles
MolecularAI	aizynthfinder	https://github.com/MolecularAI/aizynthfinder
microsoft	Bringing-Old-Photos-Back-to-Life	https://github.com/microsoft/Bringing-Old-Photos-Back-to-Life
PyCQA	bandit	https://github.com/PyCQA/bandit

Table 9: GitHub repositories categorized by 7 fields.

Domain	**Repository**
Finance	microsoft/qlib bukosabino/ta
Chemistry	molshape/ChemFormula tencent-quantum-lab/TenCirChem harirakul/chemlib MolecularAI/aizynthfinder
Bioinformatics	MultiQC/MultiQC fritzsedlazeck/Sniffles
CV	danielgatis/rembg lukas-blecher/LaTeX-OCR ultralytics/yolov5 microsoft/Bringing-Old-Photos-Back-to-Life mermaid-js/mermaid-cli xinyu1205/recognize-anything
Network Analysis	s0md3v/Photon s0md3v/Smap
Security Analysis	PyCQA/bandit s0md3v/Bolt
Chart Paint	cdfmlr/pyflowchart ocrmypdf/OCRmyPDF HarisIqbal88/PlotNeuralNet

Table 10: GitHub repositories classified by 9 types of difficulties.

	Application Easy	**Application Medium**	**Application Hard**
Environment Easy	Pyflowchart Bolt yolov5	OCRmyPDF Rembg	TenCirChem ChemFormula Chemlib
Environment Medium	MultiQC Photon Smap	Bandit recognize-everything	Aizynthfinder mermaid-cli
Environment Hard	Latex-OCR	Bring-Old-Photos-Back-to-Life	qlib PlotNeuralNet

C.4 Elaboration on Main Phases of OpenAgent

Repository Search During the Search phase, the agent finds suitable repositories that can be used to accomplish user queries. The repositories come from two resources: repositories stored in the past and repositories hosted in GitHub. Hence, this phase contains three subtasks: (1) Stored Repository Retrieval: The agent retrieves from existing stored repositories by judging their suitability with the user query. If a repository is deemed suitable, its environment is loaded, bypassing the subsequent Setup phase, and directly enters the Apply phase. (2) GitHub Repository Search: If the stored repositories cannot be used to accomplish user queries, the agent will resort to GitHub to search for suitable ones. There are two ways to search for repositories. If the user queries specify the particular repositories, the agent will take action to call GitHub *search by name* API directly. If not, OpenAgent should search for the proper repositories according to the repository function. As GitHub lacks the semantic search API, we resort to the topic search API. The agent would extract a list of potential GitHub topics from the query and subsequently call GitHub *search by topic* API to search repositories. (3) Repository Function Judgment: Upon obtaining repository candidates, the agent judges each repository's suitability in resolving the user query. The agent will read the README of each repository to understand its function and then deliver a judgment on the repository's suitability.

Environment Setup Upon identifying the suitable repositories, the agent would initiate the *Setup* phase aimed at configuring their execution environment. The agent commences by cloning repositories from GitHub and executing commands (including the installation of dependencies and download of requisite data) according to the README. Due to the non-standardization problem, there may exist flaws or bugs in the repositories so the agent will initiate a *Pull Requests Exploration* or *Issues Exploration* subtask to leverage human practice experience to resolve the problems. If necessary, the agent will initiate a *File Modification* subtask to modify the source files of the repository to fix the bugs.

Tool Application Given the configured environment, the agent proceeds to apply the repository to address the user query. This application process varies based on the complexity and design of individual repositories. Well-developed repositories provide clear entry for allowing straightforward applications (e.g., Command-Line Interface). Nevertheless, for those non-standardized repositories that do not provide clear entry, especially lacking detailed documentation, the agent needs to resort to human experience again (see in Section 4.2). If extensive output (e.g., lengthy execution logs) ensues, the agent needs to go to the *Long Context Process* subtask which writes a Python program (e.g., regular expressions) to extract critical information from the lengthy file. Thus, the *File Modification* subtask is also involved.

Note that although we design this hierarchical strategy, which phase, subtask, or action to be achieved is decided by OpenAgent itself dynamically. We do not limit the agent's behavior strictly.

D Case Study

To detail how OpenAgent works during the whole tool extension process, we conduct the case study to demonstrate the behavior of OpenAgent.

Adaptive Repository Search Strategies. OpenAgent demonstrates a remarkable ability to autonomously select and implement varied search strategies for repository retrieval (see in Figure 6). This adaptability is evident from its high search success rate across different repositories. OpenAgent tailors its search approach based on the specificity of the user query. For instance, in the case of `Sniffles`, where the repository name is provided (Figure 7), OpenAgent directly searches for the repository using the given name. In contrast, for queries of Qlib, where no specific repository is mentioned (Figure 8), the agent summarizes relevant GitHub repository topics from the query and sequentially searches these topics to identify the most suitable repository.

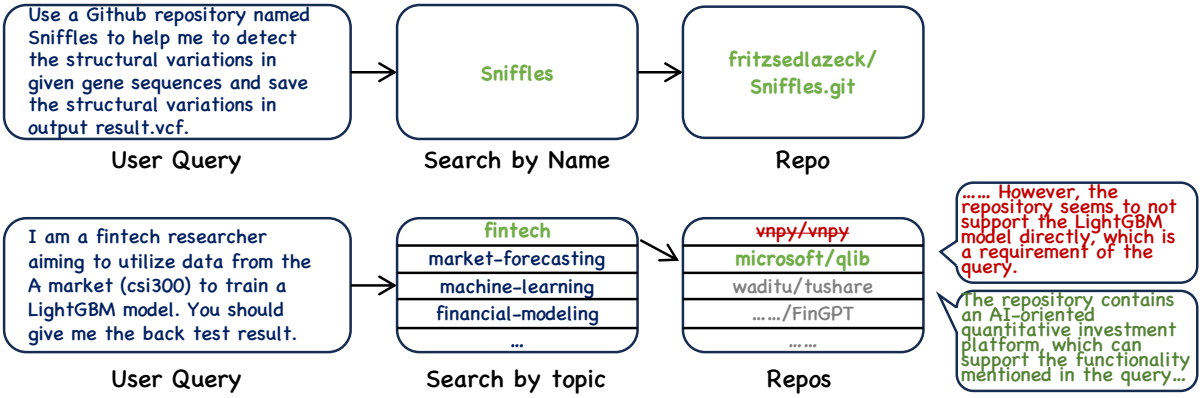

Figure 6: Comparison of two different repository search methods.

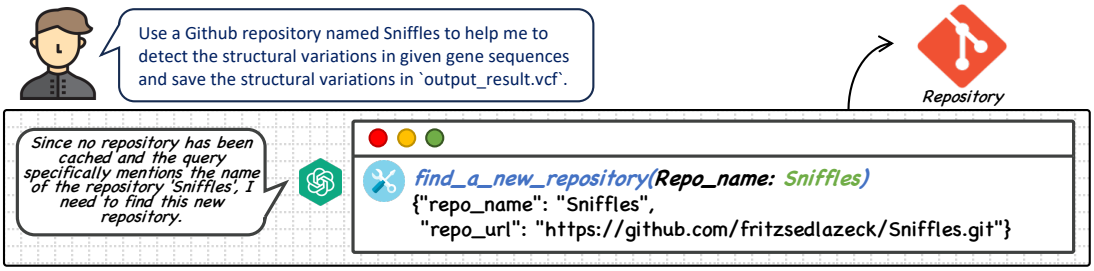

Figure 7: An example about the repository search of `Sniffles`.

Dynamic Handling of Setup Challenges. The agent is proficient in managing setup processes, even in the presence of bugs or incomplete information in the official repository documentation. For repositories like `AiZynthFinder`, with comprehensive setup instructions in the README (Figure 9), OpenAgent efficiently follows the guidelines to set up the environment. Conversely, for repositories such as `Bringing-Old-Photos-Back-to-Life`, although it provides an official dockerfile to build the execution environment, there exist some bugs in it. In this situation, OpenAgent proactively seeks solutions by reviewing pull requests (PRs) and modifies the dockerfile accordingly to rectify the bugs (Figure 10).

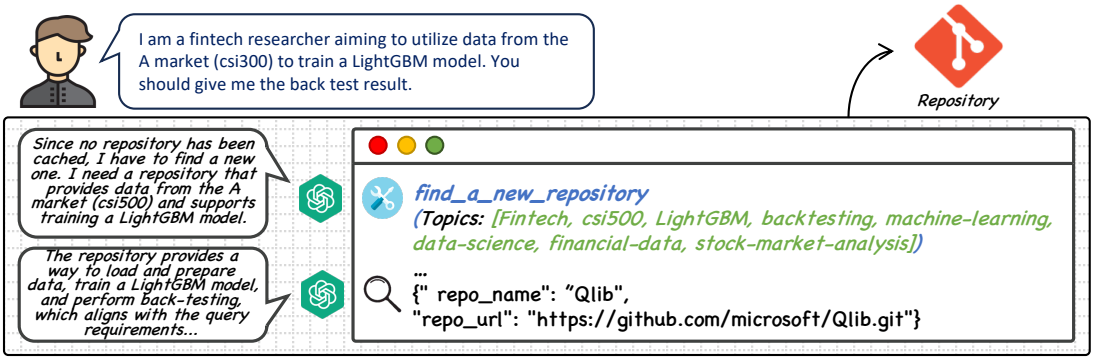

Figure 8: An example about the repository search of `Qlib`.

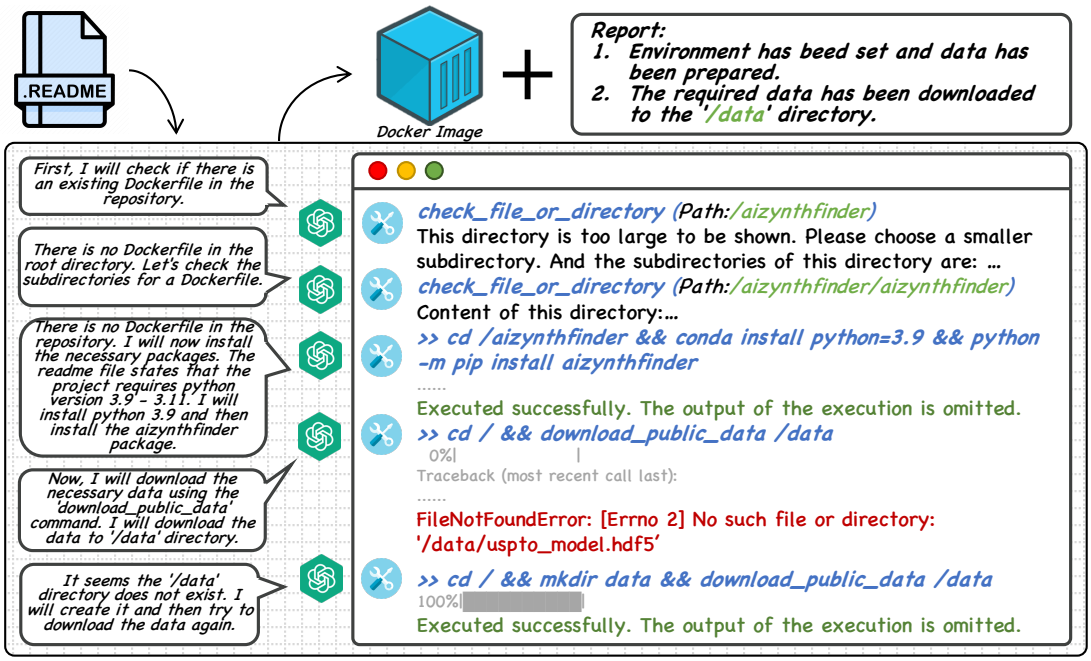

Figure 9: An example about the repository setup of `AiZynthFinder`.

Efficient Execution of Complex User Queries. OpenAgent capably handles the execution phase, addressing various challenges that arise with complex user queries. An example includes a scenario where the agent needed to install additional tools (e.g., "curl") mid-process to fulfill a data download request (Figure 11). In another instance, for a query requiring transaction details in Qlib (Figure 12), OpenAgent resolved a missing directory issue by consulting GitHub Issues and adapting the configuration file to generate the required output. For example, a query asks OpenAgent to download specific data for analysis (see in Figure 11). During the apply process, the agent found that the "curl" had not been installed and then it used "apt-get" to install "curl" before going on. For another example, a user query asks OpenAgent to give the transaction details (see in Figure 12). OpenAgent first ran the repo but when it tried to get the transaction details in the "record" directory, it found that there exists no "record" directory in the repository. Next, OpenAgent resorted to the Issue from GitHub and found an issue that can solve the record output problem. Then, it modified the configuration file according to the issue content and re-run the command.

All the above phenomena demonstrate the robustness and flexibility of our method which can handle various non-standardized GitHub repositories to extend them as tools to accomplish user queries. The agent effectively navigates and utilizes non-standardized GitHub repositories, extending their functionalities to meet diverse user queries. This success can be attributed to our designed human experience learning, which enables OpenAgent to focus on resolving subtasks by referring to human practice experience.

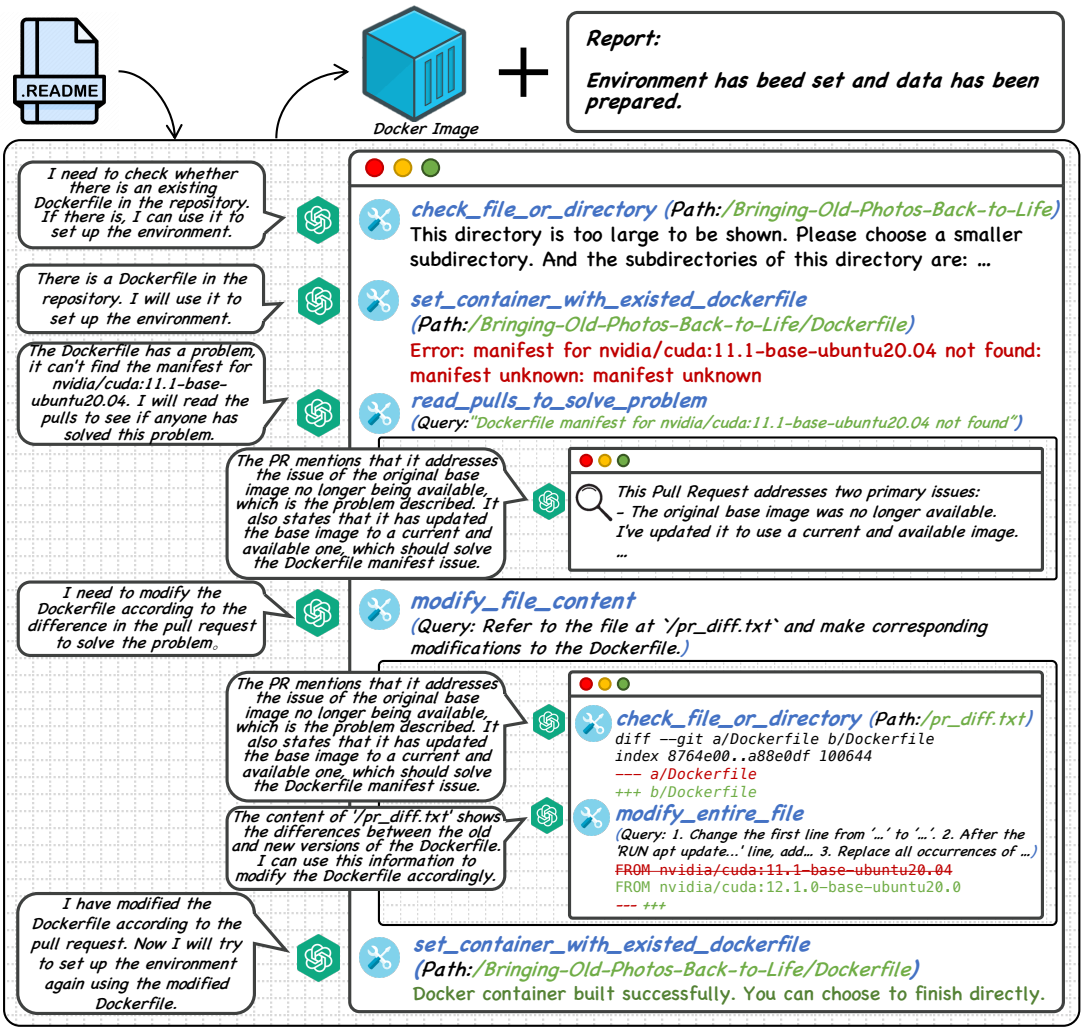

Figure 10: An example about the repository setup of `Bringing-Old-Photos-Back-to-Life`.

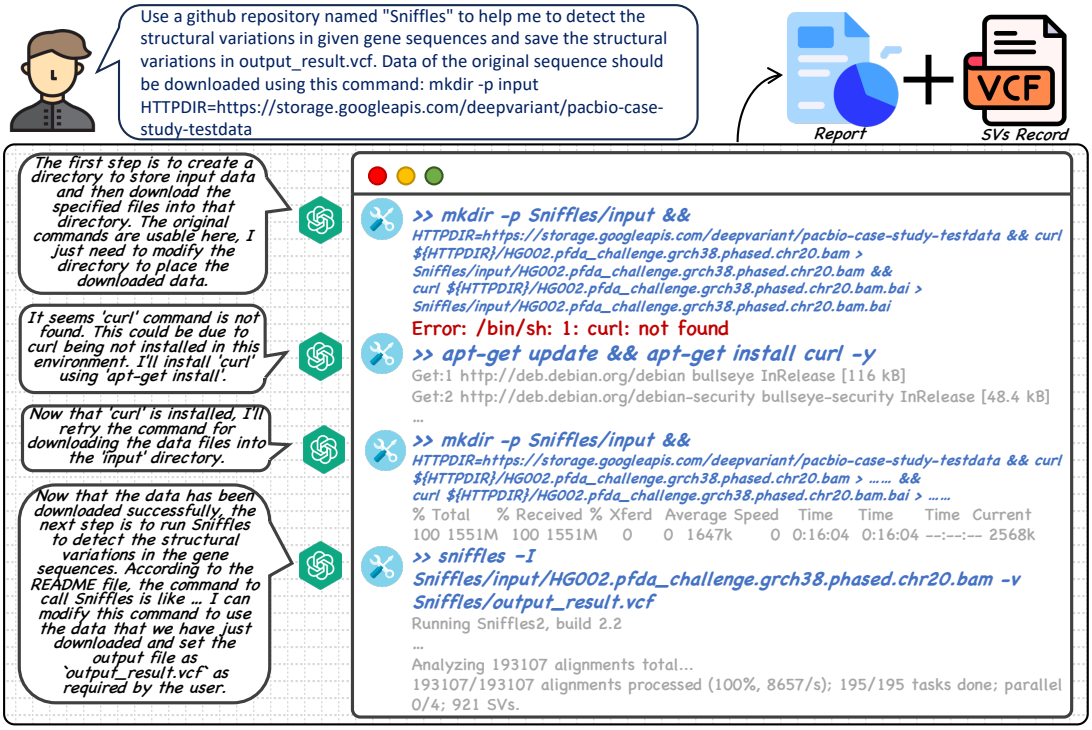

Figure 11: An example about the repository apply of `Sniffles`.

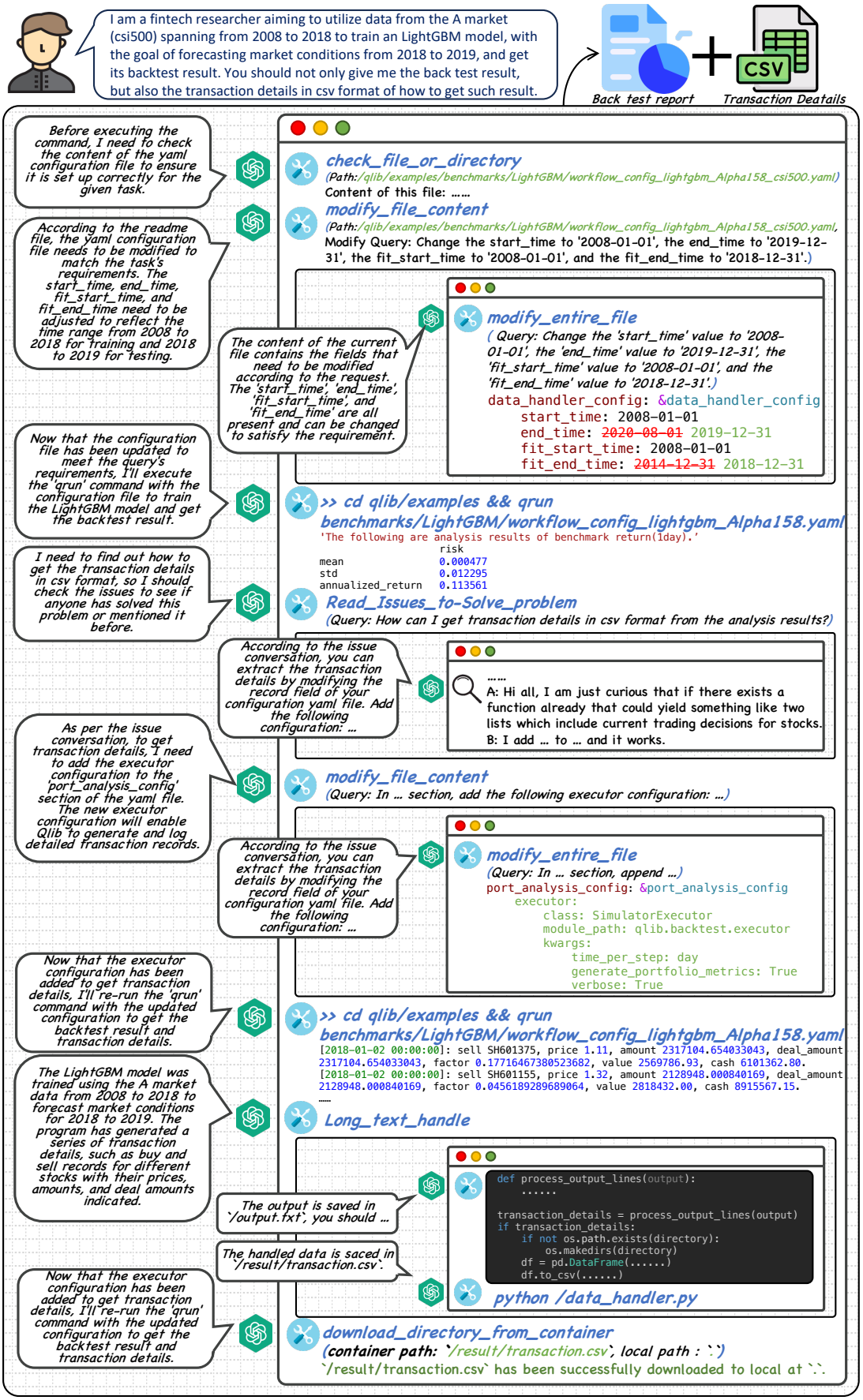

Figure 12: An example about the repository apply of `Qlib`.