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

Anonymous submission

Abstract

Large Language Models (LLMs) excel in natural language processing but struggle with complex problem-solving in diverse technical domains, particularly those requiring precise calculations or complex simulations. While connecting LLMs to external tools to build LLM-based Agents can enhance their capabilities, existing approaches often 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 diverse domain knowledge that requires tools to solve. To address this gap, we introduce **OpenAct**, a dataset comprising 339 science questions spanning 7 diverse domains that need to be solved with domain-specific methods In our experiment, even state-of-the-art LLMs and LLM-based Agents like GPT-4 and XAgent demonstrate shallow success rates on OpenAct, underscoring the need for a novel approach. In response, we present **OpenAgent**, an innovative LLM-based Agent that can tackle evolving queries in open domains, through autonomously integrating specialized tools. OpenAgent employs a novel hierarchical LLM framework, Agency, where specialized agents handle specific tasks. OpenAgent integrates new tools through a four-phase process, learning from human discussions to overcome integration challenges and enhance its capabilities. Evaluation on OpenAct demonstrates OpenAgent's superior effectiveness and efficiency that significantly outperforms current methods.

Introduction

Large Language Models (LLMs) (OpenAI 2022, 2023) have demonstrated exceptional capabilities through diverse kinds of traditional natural language processing (NLP) tasks. However, LLMs still struggle with specialized tasks that require calculation, simulation, data augmentation, etc. To tackle it, researchers equip LLMs with external tools (e.g., search engines (Nakano et al. 2021; Qin et al. 2023a), calculators (Schick et al. 2023)) to function as agents that are capable of performing complex tasks, thus extend the capability boundary of LLMs beyond traditional NLP tasks. Existing LLM-based agents (AutoGPT 2023; Wu et al. 2023; XAgent 2023; Nakano et al. 2021; Qin et al. 2023a; Schick et al. 2023; Parisi, Zhao, and Fiedel 2022) have access to a pre-defined toolset, effectively combining the LLM's cognitive abilities with the specialized functionalities of these tools.

However, the effectiveness of current LLM Agents is constrained by the predefined toolset they rely on. This constraint restricts them to addressing only in-domain problems, being incapable of solving diverse, open-domain questions. Therefore, these agents don't have the generalization ability on a domain level. Moreover, the benchmarks used to demonstrate these agents' effectiveness are constructed based on these predefined toolsets rather than real-world demands, which fail to meet the requirements of open-domain tasks that often require tools beyond the established set.

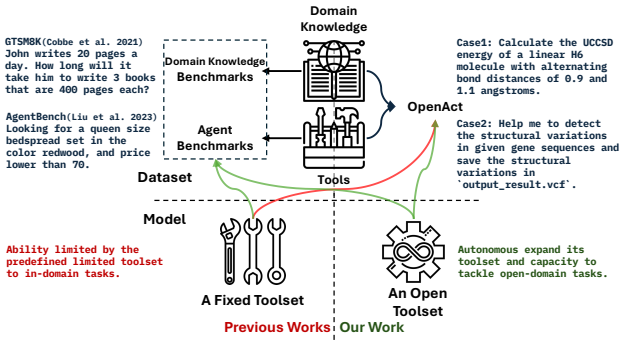

Figure 1: The comparison between our work and previous studies is illustrated from both dataset and model perspectives. ↗ indicates that a model is capable of solving tasks within a benchmark, whereas ↗ signifies that it is unable to do so.

Numerous real-world tasks necessitate specialized tools and domain-specific knowledge that extend beyond the inherent capabilities of pre-trained language models and a predefined 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. For large language models to effectively address these specialized tasks, they require access to such tools, preferably through command-line interfaces (CLIs) or application programming interfaces (APIs). In this context, GitHub emerges as a valuable resource as it contains state-of-the-art implementations of algorithms and methodologies employed by experts in their respective fields.

GitHub repositories not only reflect cutting-edge technologies across various disciplines but also highlight the important problems and research directions in these fields. Based on this understanding, we introduce OpenAct. The construction

of OpenAct began by identifying key issues and methodologies across multiple specialized domains. We then referenced relevant tools and implementations on GitHub to carefully design a series of tasks that both reflect actual domain needs and fall within the potential capabilities of large language models. OpenAct is the first comprehensive dataset designed to evaluate LLMs on fulfilling open-domain real-world tasks. It comprises 339 queries spanning 7 diverse domains, including finance, chemistry, bioinformatics, computer vision, and others. These queries are meticulously curated to require specialized knowledge and multi-step problem-solving approaches, extending far beyond the capabilities of standard LLM Agents with predefined toolsets. Selected tasks in traditional benchmarks and ours are list in Table 1.

To address this challenge, we propose a novel approach: empowering LLM Agents to autonomously integrate tools and enhance their capabilities, rather than relying solely on a predefined toolset. This strategy aims to equip LLM Agents with the ability to adapt and expand their functionalities in response to diverse and evolving user needs.

As GitHub repositories serve as a valuable resource, 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.

For both LLMs and humans, the extension of tools based on GitHub repositories presents several challenges: (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 user's needs better. The extensive size of the repository can make it challenging to locate the portions of the code needing modification. (3) **Complex Workflow for Tool Integration**: Searching for a suitable tool setting up the environment, and utilizing the tool involve dozens of different tasks. The significant differences between these tasks can easily distract the LLMs from completing the whole process effectively.

Motivated by the feature of these challenges, we introduce **OpenAgent**, a novel LLM-based agent system that autonomously extends tools from GitHub by hierarchically decomposing the tasks for the tool extension. OpenAgent operates through several phases, starting with searching suitable repositories, then setting up the necessary environment, utilizing the repository to fulfill user queries, and finally storing the repository for efficient future use. By decomposing the tool integration process into these structured phases, OpenAgent ensures that each step is handled comprehensively. This approach not only addresses the current limitations but also equips LLM agents with the ability to autonomously adapt and grow in response to evolving user needs.

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.

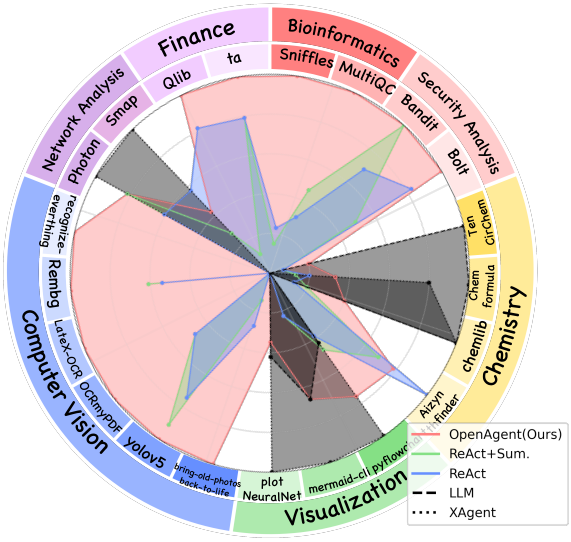

Figure 2: Illustration of GPT-4-based OpenAgent performs against baselines on 339 queries from 7 domains in OpenAct.

- We propose OpenAgent, a novel LLM-based Agent system that autonomously extends its toolset by integrating repositories from GitHub. OpenAgent employs a hierarchical structure that dynamically decompose the tool integration process into distinct phases, addressing challenges such as quality assurance and workflow complexity.
- We conduct extensive experiments on OpenAct to demonstrate the effectiveness of OpenAgent compared to state-of-the-art baselines, as demonstrated in Figure 2.

Related Work

LLM-based Agents Recent advancements in artificial intelligence have been significantly marked by the development of Large Language Models (LLMs) such as ChatGPT (OpenAI 2022), GPT-4 (OpenAI 2023), LLaMA (Touvron et al. 2023a,b), which have demonstrated remarkable proficiency across a diverse range of tasks. Benefiting from LLMs, LLM-based agents (AutoGPT 2023; Wu et al. 2023; Li et al. 2023; XAgent 2023) have attracted research attention, which aims to extend the capabilities of LLMs to interact with external tools for accomplishing real-world tasks. 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. 2023; Qian et al. 2023b; Wang et al. 2023; Qian et al. 2023a) for agents to dynamically create tools. Nevertheless, the functionalities of these created tools remain simple and limited, insufficient to meet complex real-world user queries.

Benchmarking LLMs on Domain Knowledge and Tool Use Different benchmarks evaluate LLMs across diverse domains and capabilities. Domain knowledge benchmarks initially focused on mathematics, with seminal works like

Benchmark	Num. of Domains	Task Source	Task Types	Multimodality	Code Use	Tool Use	Open End	Repository-Level
ToolBench (Qin et al. 2023b)	-	Tool	QA	✗	✗	✓	✗	✗
MetaTool (Huang et al. 2024)	-	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)
OpenAct (Ours)	7	Domain and Github	QA and Coding	✓	✓	✓	✓	✓ (21)

Table 1: Comparison of Benchmarks for Evaluating LLMs on Domain Knowledge and Tool Utilization. The "Num. of Domains" column indicates the number of knowledge domains evaluated by each benchmark, with "-" denoting benchmarks that do not assess domain knowledge. "Open-Ended" 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.

GSM8K (Cobbe et al. 2021) and MATH (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. SciBench (Wang et al. 2024) further advanced this approach by incorporating code interpreter functionality while maintaining 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. 2024; Liu et al. 2023) are generally designed based on the functionalities of various tools and APIs. More recently, benchmarks 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(software engineering for SWE-Bench (Jimenez et al. 2024) and machine learning for ML-Bench (Tang 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.

Dataset: OpenAct

Dataset Construction

We introduce a high-quality benchmark, OpenAct, which spans 7 distinct expertise domains, that bridges knowledge in open domains with practical implementation resources.

Firstly, we engaged with specialists across 7 diverse domains to identify frontier problems in their fields that are potentially solvable through computational models. Then, we conducted an extensive search of GitHub repositories, seeking implementations that align with the identified domain challenges. This process yielded an initial pool of potential repositories for each domain. Initially, 10 candidate repositories are identified per domain. We then filter out repositories with analogous task types or similar execution methods, resulting in a refined selection of 21 repositories. Then we employ GPT-4 to generate 30 query candidates of different levels per repository. Through manual testing, we retain 5-10 queries per repository, ensuring their solvability using the corresponding repository. Finally, it results in a total of 113 high-quality queries.

To investigate the impact of different levels of repository information hints, we design three types of prompts for each query: (1) *Explicit Hint*: Specifying an available repository and its GitHub address; (2) *Implicit Hint*: Providing keywords related to the repository's domain or functionality; (3) *No Hint*: No additional prompt is hinted the user queries.

Concatenating these prompts with the generated queries yields a dataset of 339 instructions, covering diverse real-world scenarios. Ground truth answers are constructed by human experts for each query to establish a robust evaluation. The key statistics of OpenAct are presented in Table 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 three 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 need agents to find relevant Issue/PRs to solve.

Similarly, for the Apply difficulty, we divided repositories into three classes: *Apply-Easy*: Simply requires running some commands given by the README. *Apply-Medium*: Requires writing configuration files or downloading extra resources,

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

Table 2: Statistics of OpenAct.

like data and trained models. *Apply-Hard*: Requires modifying the source code of the repositories. Sometimes need to refer to relevant Issue/PRs for help.

Local Repository Snapshot

Given the dynamic nature of GitHub repositories, which can undergo significant changes over time, we create a local snapshot of GitHub repositories accessed during the experiments to ensure reproducibility and consistency. This snapshot preserves the exact state of the repositories at the time of our experiments. For repositories not included in the snapshot but used by the LLMs, we allow real-time access from GitHub, as they do not affect the controlled variables in this study.

Evaluation Metrics

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

Completeness To precisely evaluate the performance of OpenAgent under different settings, we 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's correctness against a "golden answer" from generated by human expert. These scores are subsequently aggregated and normalized to a 10-point scale to derive the overall completeness score.

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. For OpenAgent, a query is considered to "pass" when its completeness score exceeds 0.9. In scenarios where only end-to-end results are available, the evaluation is conducted exclusively based on the comparison of the final answer with the expert-generated "golden answers". A query is deemed to pass if there is a concordance between these two answers.

We sampled 120 queries and results for human annotation, achieving an 87.5% agreement with GPT-4 evaluations, indicating the high reliability of both metrics.

Methodology

The complexity of autonomously integrating tools from GitHub poses significant challenges for traditional LLM-based agents like ReAct. To address this, we propose OpenAgent, a hierarchical framework designed to effectively manage these complexities.

Our methodology consists of three key components: an Agency System for task decomposition, an Autonomous Integration process for tool integration, and an Experience Learning mechanism for continuous improvement.

Agency System

In the LLM as Agency framework, a complex task is decomposed into a hierarchical structure wherein each agent receives a query from a superior entity (whether superior agent or human) and responds by directly interacting with the world or designating sub-agent engagements. We denote them as *Action Call* and *Agent Call* respectively. Action Calls include command line execution, file check, submit to finish, etc. Agent Calls include Environment Setup, File Creation/-Modification, etc. This process can be formalized as follows:

$$A_i^n = \text{Agent}_k^n(Q^n, A_1^n, O_1^n, ..., A_{i-1}^n, O_{i-1}^n) \quad (1)$$

Here, Agent_k^n represents the k-th agent at level n of the hierarchy, A_i^n denotes the i-th action or sub-agent call by Agent^n, Q^n is the query received from the agent's superior, and O_j^n and A_j^n are respectively the observations and preceding actions/sub-agent calls that lead up to A_i^n.

If A_i^n is a sub-agent call, the query for Agent^{n+1} is derived from A_i^n, such that $Q^{n+1} \leftarrow A_i^n$. The sub-agent then executes its designated operations based on this new query.

If A_i^n constitutes an action call, two scenarios arise. If the action type is "Submit", indicating task completion, the action sequence terminates and the outcomes are reported back to Agent^n, with $O_k^{n-1} \leftarrow A_i^n$. Alternatively, the action directly interacts with the environment E, and the resultant environmental feedback is captured as O_i^n:

$$E', O_i^n \leftarrow \text{Action}(E, A_i^n) \quad (2)$$

where E' represents the updated environment after the action A_i^n is performed, and Action is an action.

This recursive process continues until the sub-tasks are reduced to atomic, non-intelligent actions. The hierarchical organization of agents and actions allows for efficient decomposition and execution of complex tasks, with each agent focusing on its specific sub-task. The collective behavior of these agents and actions gives rise to the overall intelligent behavior required for the task at hand.

We present the following pseudocode to illustrate the Agency Algorithm with five variables: Q represents the query, E represents the environment state, H represents the interaction history, A represents an action or agent, and O represents the observation. $A.Query$ means A's query for the designated inferior agent and $A.Report$ means A's report to its superior agent when finishing its tasks.

Algorithm 1: Agency Algorithm

```
Function Agent(Q, E):
    H ← [Q]
    while True do
        A ← LLM(H)
        if IsAgent(A) then
            // Call this function
               recursively
            E, O ← Agent(A.Query, E)
        else
            if A.Type = "Submit" then
                return E, A.Report
            end
            E, O ← Action(A.Action, E)
        end
        H.Append(A, O)
    end
return
```

Autonomous Integration

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 15). 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.

Knowledge Store After accomplishing the user query, the agent proceeds to store the repository together with its execution environment to facilitate future usage, especially for recurring or similar user queries. Specifically, the execution environment will be stored in a docker image so if a similar query comes, the agent can retrieve this repository and restore it to apply directly. To enhance the stored repository retrieval, the agent should abstract its understanding of the repository's functionalities (i.e., the *Function Description* subtask) and summarize the experience in the apply phase (i.e., the *Experience Summarization* subtask).

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.

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 can pose challenges. In such cases, learning from human experiences becomes an efficient approach. Building upon the Agency System framework, we introduce a specialized agent, the Issue/PR Agent, $\text{Agent}_{\text{Issue/PR}}$, 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. $\text{Agent}_{\text{Issue/PR}}$ is responsible for searching, evaluating, and returning relevant information from GitHub Issues and Pull Requests.

We can formalize the Issue/PR Agent's role within the Agency framework as follows:

$$\text{Agent}_{\text{Issue/PR}}(Q^{'}) \leftarrow A_i^n(H) \tag{3}$$

where H is the interaction history, $Q^{'}$ is the query for $\text{Agent}_{\text{Issue/PR}}$ to solve. $\text{Agent}_{\text{Issue/PR}}$ will then retrieve a set of Issus/PRs $S = \{s_1, s_2, ..., s_n\}$ from GitHub, judge them one by one:

$$r_i \leftarrow \text{Agent}_{\text{Issue/PR}}(Q^{'}, s_i) \quad i = 1, 2, \ldots, n \tag{4}$$

where r_i to r_n are relevance scores. Then it select the most relevant solution $s^* = \arg\max_{s_i \in S} r_i$ and report it to A_i^n, which will result in $H \leftarrow H + [\text{Agent}_{\text{Issue/PR}}, s^*]$. A_i^n will then continue with PR/Issue augmented history.

Apart from in-task knowledge learned from community experiences and solutions, OpenAgent can also learn from its own experiences across tasks, and solidify these experiences.

The agent's policy $\pi(A_t|S_t, E)$ defines the probability of taking action A_t given the state S_t and accumulated experience E. S_t includes current observations and historical information, while E represents past learning outcomes. The policy aims to maximize expected long-term rewards. If no prior experience E exists, the policy is $\pi(A_t|S_t)$.

After each task, the agent summarizes its experience based on the environmental feedback, formalized as a set of rules or patterns E.

$$E \leftarrow f(E, R_t, S_t, A_t) \tag{5}$$

where $f(\cdot)$ is the experience summarization function, implemented using a large language model. R_t is the semantic reward obtained after action A_t.

Methods	Finance	Chemistry	Bioinformatics	Computer Vision	Network Analysis	Security Analysis	Visualization	All
GPT-3.5-Turbo Based								
LLM	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								
LLM	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.

If there is no prior experience E, the summarized self-experience E is added to the decision process, updating the policy $\pi(A_t|S_t)$ to $\pi(A_t|S_t, E)$. If E exists but changes, $\pi(A_t|S_t, E)$ adapts accordingly. This process introduces an experience-based heuristic method, allowing the agent to reference past experiences and make more informed decisions, ultimately improving task success rates.

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R_t|S_t, A_t, \pi] \tag{6}$$

Through this adaptive and evolving approach, the agent not only enhances its ability to solve specific tasks but also develops a general pathway for learning and improvement, maintaining efficiency and adaptability in diverse and dynamically changing task environments.

Experiment

Experiment Settings

Baseline To validate the effectiveness of our OpenAgent, we design the following baselines: (1) LLM: Raw LLMs without external tools (2) ReAct (Yao et al. 2022): ReAct is a widely-used LLM-based agent task-solving technique (Auto-GPT 2023; Wu et al. 2023). It can accomplish intricate tasks on the fly by decomposing them into explicit intermediate steps. 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. (5) XAgent (XAgent 2023): XAgent is a powerful general-purposed LLM-based agent, which 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. All actions including API callings, command executions, and agent operations are implemented based on the Function Calling feature[1] of GPT series. To denote the finish of each phase (Search, Setup, Apply, Store), we additionally add *Submit* action in the function list. If the agent thinks it has accomplished each phase, it should call the *Submit* function to finish the phase. Additionally, the OpenAgent results presented here reflect the success rate on the first encounter with the problem, incorporating PR/Issue but without using experience summary to enhance the strategy.

Overall Evaluation

Table 3 reports the Pass Rates of each method. We get several observations: (1) While raw 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 tool extension is a challenging task needing powerful LLMs to achieve. (5) OpenAgent with agency architecture significantly outperforms all baselines and incurs the least computation cost in both settings, demonstrating the effectiveness and efficiency of the tool extension.

Impact of Human Experience Learning

Method	w/o PR/Issue	w/ PR/Issue	w/ Sum. Exp
GPT-3.5	8.2	17.1	58.8
GPT-4	40.3	47.3	82.3

Table 4: Results of Human Experience Learning.

To evaluate the effectiveness of human experience learning, we conducted an ablation study by removing the PRs and Issues availability and re-running the main experiments. The

[1]https://openai.com/blog/function-calling-and-other-api-updates

results, shown in Table 4, highlight the significant impact of utilizing PRs/Issues and summarized experience.

Without PRs/Issues, the pass rate for GPT-4 drops to 40.3%, clearly demonstrating the non-standardization problem of GitHub repositories and the necessity of learning from PRs/Issues to overcome these challenges.

Furthermore, to validate the effectiveness of the summarized experience, we utilized a GPT-4-based model to summarize the practice experience and then re-ran the queries, allowing the model to retrieve and use the stored experience.

The results show that leveraging the experience summarized by a GPT-4-based model, a GPT-3.5-based model can achieve a higher pass rate than a GPT-4-based model without summarized experience. This further proves the effectiveness and necessity of human experience learning.

Simultaneously, the GPT-4-based model achieves an even higher pass rate when utilizing the experience it summarized itself. This indicates that the model can learn from previous practical experiences to boost its performance. Such an advantage is highly beneficial for real-world applications as it means that the model will evolve along with its practice, continually improving and adapting to new challenges.

Impact of Search Difficulty

Hint Type	Explicit	Implicit	No Hint
Search Success Rate	96.3	65.5	31.9

Table 5: Analysis for the search difficulty.

Searching GitHub for proper repositories is challenging, so we conducted experiments to show the impact of search difficulty. As introduced, we designed three types of prompts to denote target repositories and calculated the Success Rate for each. If OpenAgent finds the correct repositories for a query, it is a search success (it needn't be the exact repository the query was constructed from). We then calculated the proportion of successful searches for each type of prompt, as shown in Table 5. Explicit Repo Prompt had the highest Search Success Rate (near 100%) as it specified repositories. Implicit Repo Prompt achieved 65.5%, indicating OpenAgent can infer relevant GitHub Topics from domains or careers. With no repository prompt, the search success rate dropped significantly. This shows OpenAgent struggles to infer GitHub Topics from the query alone, indicating a need for further research to improve performance in this scenario.

Impact of Setup and 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

Table 6: Analysis for the setup and apply difficulty.

Table 6 shows the Pass Rates for the repositories categorized based on Setup and Apply difficulties.

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.

Error Analysis

Despite our method's ability to autonomously extend tools from GitHub, we observe some failures.

Repository Select Failure OpenAgent sometimes selects inappropriate repositories to address user queries, especially when the query does not specify a repository. The agent's decision heavily relies on README files, which may lack clear descriptions. For instance, in the finance scenario, OpenAgent sometimes wrongly chose the `vnpy` repository, which is meant for quantitative trading rather than for research on models and strategies.

Environment Configuration Failure Failures in environment setup were noted in repositories like `Bringing-Old-Photos-Back-to-Life`, where a non-functional `dockerfile` was presented, and the correct one was in a Pull Request. OpenAgent sometimes modified the incorrect `dockerfile` instead of accessing the correct version, resulting in unresolved bugs and setup failures.

Execution Configuration Failure The `Qlib` repository requires a specific configuration file for execution, including dataset settings, model hyperparameters, and backtesting parameters. Misconfigurations, such as incorrect time ranges or file paths, led to execution failures. These errors highlight the need for more robust decision-making and improved repository documentation.

These insights indicate areas for improving OpenAgent's robustness in complex scenarios. We also conduct a Case Study in Appendix to further illustrate OpenAgent's process.

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 autonomous tool integration significantly enhance LLM capabilities, allowing them to tackle complex tasks across diverse domains. By leveraging human experience through GitHub repositories, OpenAgent adapts and improves over time, showing promise for future applications in intricate problem-solving. Our work paves the way for more robust and flexible LLM-based agents, capable of evolving alongside rapidly changing technological landscapes. Future research will focus on further refining the integration process and expanding the range of applicable domains, ultimately aiming to create a versatile agent that can seamlessly handle a wide array of real-world challenges.

References

AutoGPT. 2023. AutoGPT.

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

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. arXiv:2103.03874.

Huang, Y.; Shi, J.; Li, Y.; Fan, C.; Wu, S.; Zhang, Q.; Liu, Y.; Zhou, P.; Wan, Y.; Gong, N. Z.; and Sun, L. 2024. MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use. arXiv:2310.03128.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. R. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues? In *The Twelfth International Conference on Learning Representations*.

Li, G.; Hammoud, H. A. A. K.; Itani, H.; Khizbullin, D.; and Ghanem, B. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Scale Language Model Society. arXiv:2303.17760.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; Zhang, S.; Deng, X.; Zeng, A.; Du, Z.; Zhang, C.; Shen, S.; Zhang, T.; Su, Y.; Sun, H.; Huang, M.; Dong, Y.; and Tang, J. 2023. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688.

Lu, P.; Mishra, S.; Xia, T.; Qiu, L.; Chang, K.-W.; Zhu, S.-C.; Tafjord, O.; Clark, P.; and Kalyan, A. 2022. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. arXiv:2209.09513.

Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; Kim, C.; Hesse, C.; Jain, S.; Kosaraju, V.; Saunders, W.; 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. arXiv:2303.08774.

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

Qian, C.; Cong, X.; Yang, C.; Chen, W.; Su, Y.; Xu, J.; Liu, Z.; and Sun, M. 2023a. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

Qian, C.; Han, C.; Fung, Y. R.; Qin, Y.; Liu, Z.; and Ji, H. 2023b. CREATOR: Disentangling Abstract and Concrete Reasonings of Large Language Models through Tool Creation. *arXiv preprint arXiv:2305.14318*.

Qin, Y.; Cai, Z.; Jin, D.; Yan, L.; Liang, S.; Zhu, K.; Lin, Y.; Han, X.; Ding, N.; Wang, H.; et al. 2023a. WebCPM: Interactive Web Search for Chinese Long-form Question Answering. *arXiv preprint arXiv:2305.06849*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. 2023b. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv preprint arXiv:2307.16789*.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv preprint*, abs/2302.04761.

Sun, L.; Han, Y.; Zhao, Z.; Ma, D.; Shen, Z.; Chen, B.; Chen, L.; and Yu, K. 2023. SciEval: A Multi-Level Large Language Model Evaluation Benchmark for Scientific Research. arXiv:2308.13149.

Tang, X.; Liu, Y.; Cai, Z.; Shao, Y.; Lu, J.; Zhang, Y.; Deng, Z.; Hu, H.; An, K.; Huang, R.; Si, S.; Chen, S.; Zhao, H.; Chen, L.; Wang, Y.; Liu, T.; Jiang, Z.; Chang, B.; Fang, Y.; Qin, Y.; Zhou, W.; Zhao, Y.; Cohan, A.; and Gerstein, M. 2024. ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code. arXiv:2311.09835.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

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

Wang, X.; Hu, Z.; Lu, P.; Zhu, Y.; Zhang, J.; Subramaniam, S.; Loomba, A. R.; Zhang, S.; Sun, Y.; and Wang, W. 2024. SciBench: Evaluating College-Level Scientific Problem-Solving Abilities of Large Language Models. arXiv:2307.10635.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Zhang, S.; Zhu, E.; Li, B.; Jiang, L.; Zhang, X.; and Wang, C. 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.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2022. React: Synergizing reasoning and acting in language models. *ArXiv preprint*, abs/2210.03629.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) [YES]
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) [YES]
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper (yes/no) [YES]

Does this paper make theoretical contributions? (yes/no) [YES]
If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no) [YES]
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no) [YES]
- Proofs of all novel claims are included. (yes/partial/no) [YES]
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no) [YES]
- Appropriate citations to theoretical tools used are given. (yes/partial/no) [YES]
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA) [YES]
- All experimental code used to eliminate or disprove claims is included. (yes/no/NA) [YES]

Does this paper rely on one or more datasets? (yes/no) [YES]
If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) [YES]
- All novel datasets introduced in this paper are included in a data appendix. (yes/partial/no/NA) [YES]
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no/NA) [YES]
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes/no/NA) [YES]
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes/partial/no/NA) [YES]
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing. (yes/partial/no/NA) [NA]

Does this paper include computational experiments? (yes/no) [YES]
If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. (yes/partial/no). [YES]
- All source code required for conducting and analyzing the experiments is included in a code appendix. (yes/partial/no) [YES]
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no) [YES]
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no) [YES]
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (yes/partial/no/NA) [NA]
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (yes/partial/no) [NA]
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (yes/partial/no) [YES]
- This paper states the number of algorithm runs used to compute each reported result. (yes/no) [YES]
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (yes/no) [YES]
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (yes/partial/no) [YES]
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (yes/partial/no/NA) [YES]
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (yes/partial/no/NA) [NA]

Appendix

This Appendix includes:

- Prompts for OpenAgent
- Details of OpenAct
- Case Study of OpenAgent on OpenAct

Prompts

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 the `
apply` function well completes the query. If
 it lacks 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 the `search_by_query`
function to find another repository if you
think this repository is not suitable for
your query.

Query:{{query}}
```

Search Agent

w/o Cached Repositories

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

w/ Cached Repositories

```
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 solve your
task, call the `use_existing_repository`
function to use it. Otherwise, call the `
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 existing
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}}

......
```

Setup Agent

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

You will be provided with the readme file of
 the repository. You can also use the `
check_file_or_directory` function to check
the `/<==repo_name==>` directory whether
there is an existing Dockerfile. If setting
up the environment is complex and there is
an existing dockerfile, you can use the `
set_container_with_existed_dockerfile`
function to directly use that dockerfile. If
 there is any problem with the dockerfile,
you can try to use the `
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 existing
dockerfile is built successfully, you can
call the `submit` function directly with the
 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 tries, it means there isn't a
dockerfile in this repository. You don't
need to try more times.

If there is no existing docker file, you
should analyze the readme file 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 'continuum/miniconda3'.
 If an error happens due to an inappropriate
 base image, you can use `echo` to create a
docker file yourself, with the 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 has been cloned 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 a problem with the version of Python, please reinstall Python of the appropriate version with `conda install python=<version>`.
- If a function you called returns you with a file path, you should pass the file path to the next function you call if needed.
- If there are different choices to do the same task and you fail to use 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.

Readme of the repository: {{readme}}.

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 prepare the data, or check relevant files about the environment or data anymore unless the user provides 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 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 results 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 are any special demands in the query, you should check the final executable file to see 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 lacks of required element(s), which may be because the repository itself doesn't support the 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 first use `upload_directory_to_container` to upload it from local to the docker container. By default, the path claimed in the query is local, you need to upload it. If a required message can be retrieved from the output of the execution of the program, summarize it in 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 it if necessary. You should also ensure that the required directories all exist before running a program.

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

Query: {{query}}

Readme:{{readme}}

Modify Agent

You are a professional programmer. Your task is to modify (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 by modifying this file, you can modify this file.

If the query contains a 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 messages by checking and summarizing, just pass the query path and let the "modify_entire_file" function 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 a relative, 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 the `/opt/conda/lib/python3.11/
site-packages/<package name>` directory (
python version should be decided by using
the `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.

Modification Query: {{query}}

Original Code: {{code}}
```

Judge Agent

```
You are a professional programmer. Your task
 is to judge how well a programmer uses a
GitHub repository to handle a query. You
will be given a query and the actions the
programmer took to handle the query. If the
task includes an input or output file, you
will be given a path to the programmer's
output. The input path is in the query and
the 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 the directory or file
saved locally. If no input path, output path
, or truth path is given, do not check the
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 the user-calling functions to use the
 repository. For correctly setting up the
environment and preparing the data, 2 points
 should be added for the environment and 1
point should be added for the data. If no
data is required, a point for data should be
 added.\ In 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
instructions in the readme. If the right
actions(including commands and function
calling) are taken and get a result, you
should add 4. If the 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 the user calling functions to use the
 repository, without the steps the
environment is set up .\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 instructions in the readme. If
the right actions(including commands and
function calling) are taken and get a result
, you should add 4. If the 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 a 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}}
```

Repository Details

Overall Statistics

Please refer to Table 7 for the overall statistics regarding the repositories.

Repositories Categorized by Field

Please refer to Table 8 for the field of each Github repository.

Repositories Categorized by Difficulties

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

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 3). 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 4), OpenAgent directly searches for the repository using the given name. In contrast, for queries of Qlib, where no specific repository is mentioned

Table 7: 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 8: 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

(Figure 5), the agent summarizes relevant GitHub repository topics from the query and sequentially searches these topics to identify the most suitable repository.

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 6), 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 environ-

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

	Application		
Easy	**ApplicationMedium**	**ApplicationHard**	**EnvironmentEasy** PyflowchartBoltyolov5 OCRmyPDFRembg TenCirChemChemFormulaChemlib
EnvironmentMedium MultiQCPhotonSmap Banditrecognize-everything			
Aizynthfindermermaid-cli			
EnvironmentHard Latex-OCR Bring-Old-Photos-Back-to-Life qlibPlotNeuralNet			

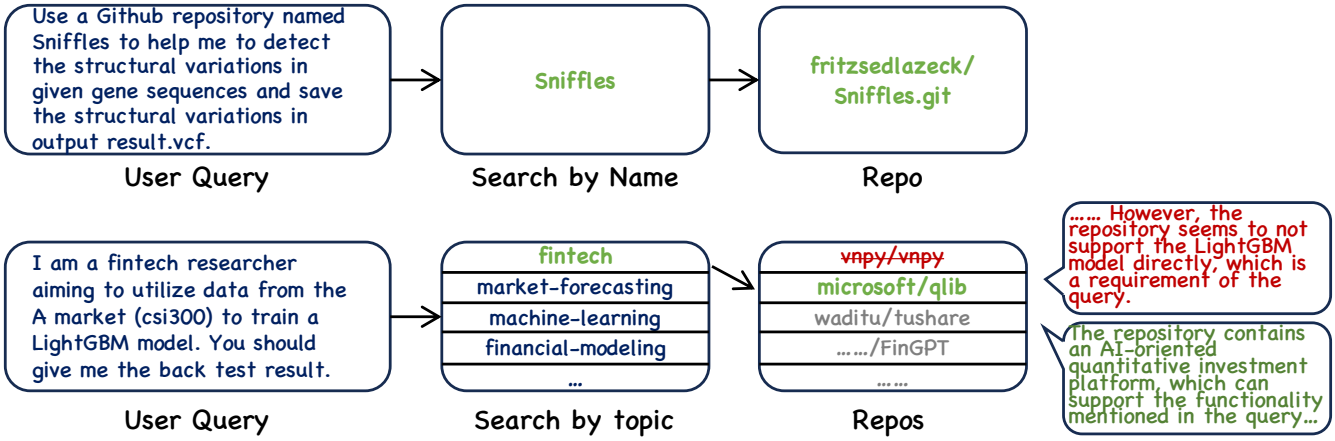

Figure 3: Comparison of two different repository search methods.

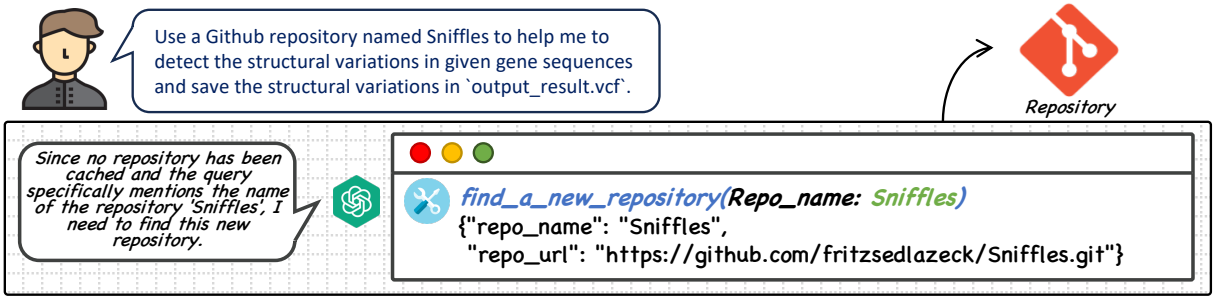

Figure 4: An example about the repository search of `Sniffles`.

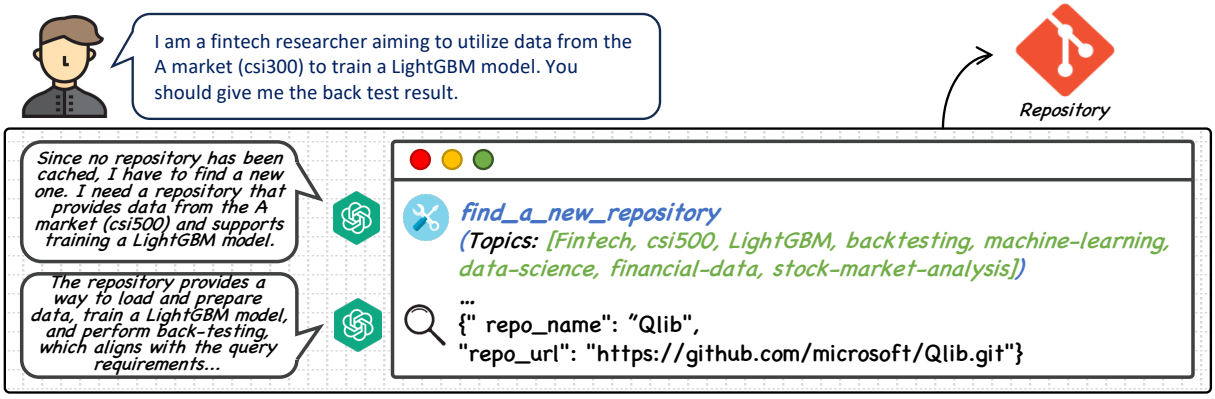

Figure 5: An example about the repository search of `Qlib`.

ment, 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 7).

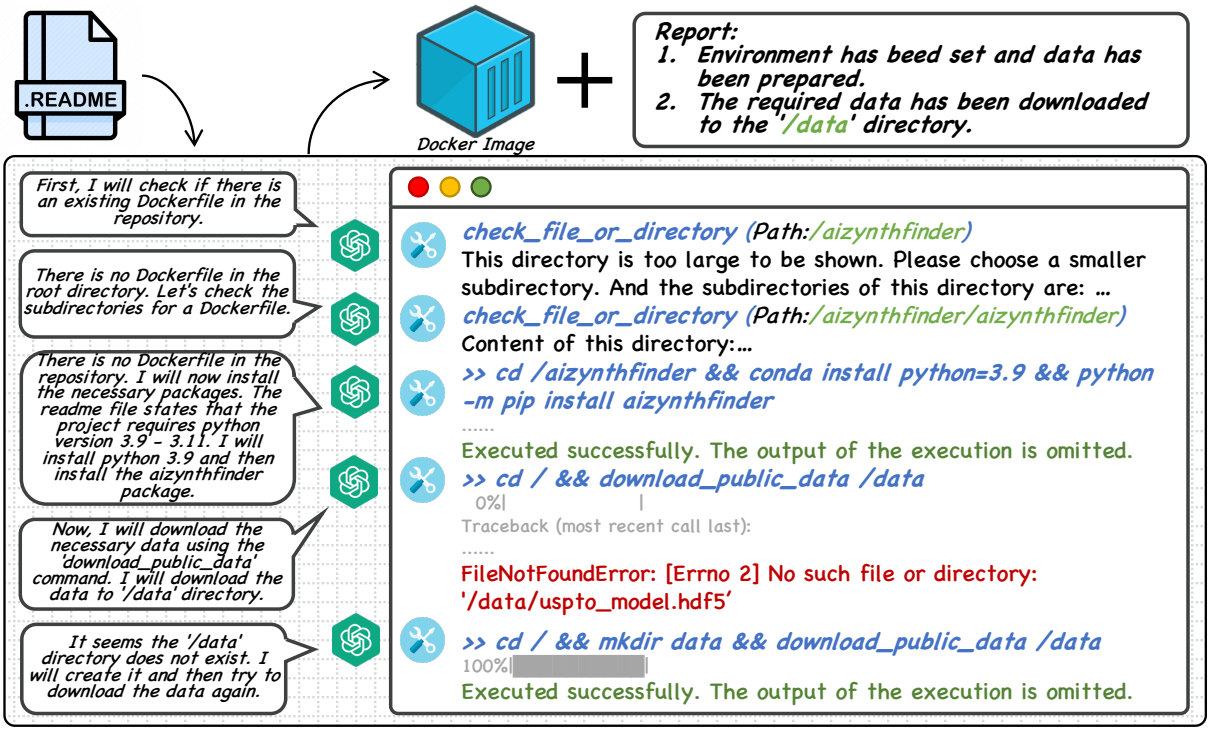

Figure 6: An example about the repository setup of `AiZynthFinder`.

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 8). In another instance, for a query requiring transaction details in Qlib (Figure 9), 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 8). 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 9). 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.

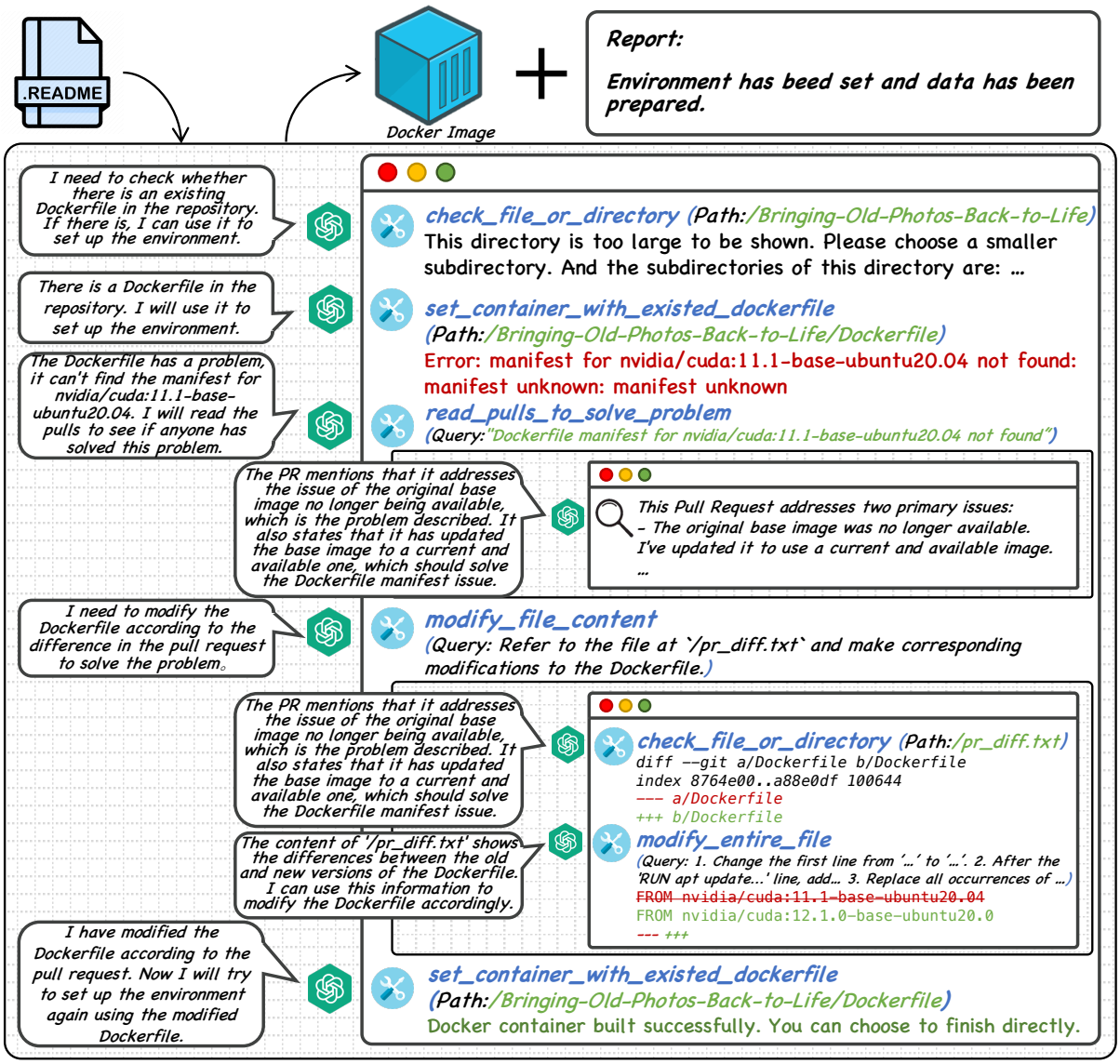

Figure 7: An example about the repository setup of `Bringing-Old-Photos-Back-to-Life`.

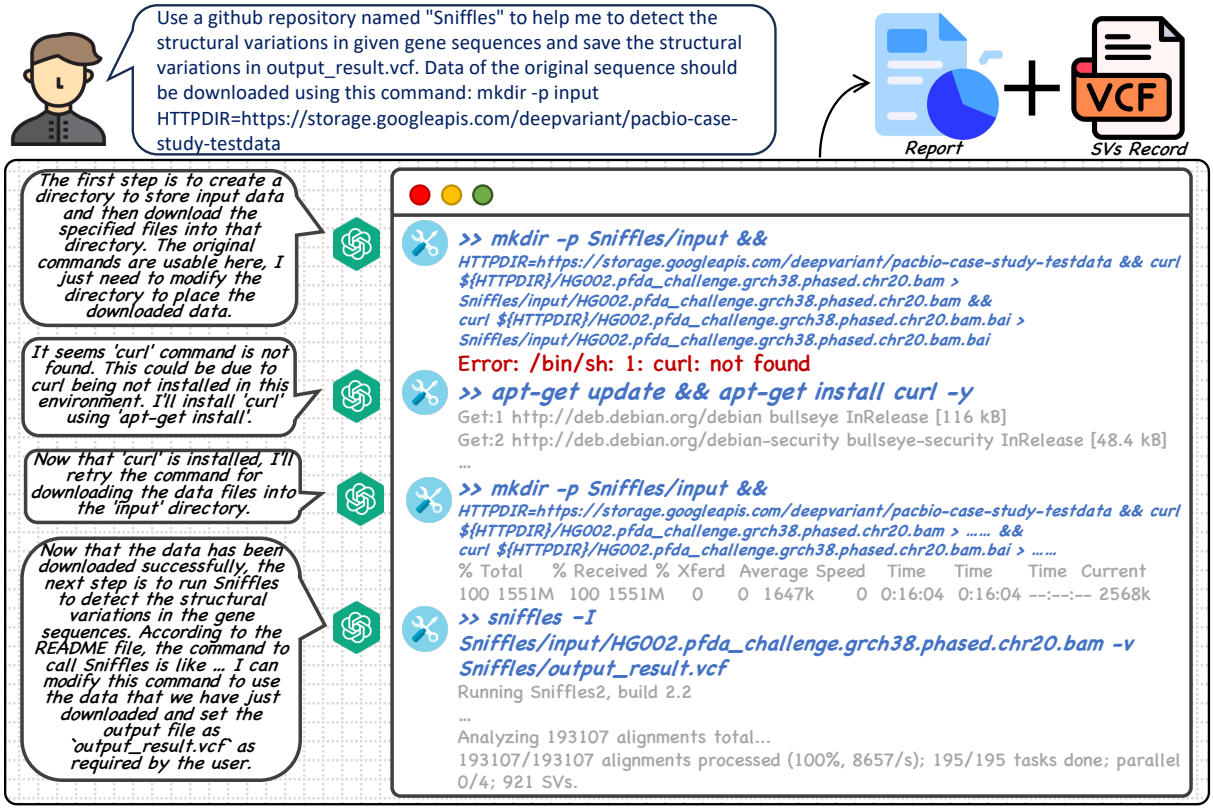

Figure 8: An example about the repository apply of `Sniffles`.

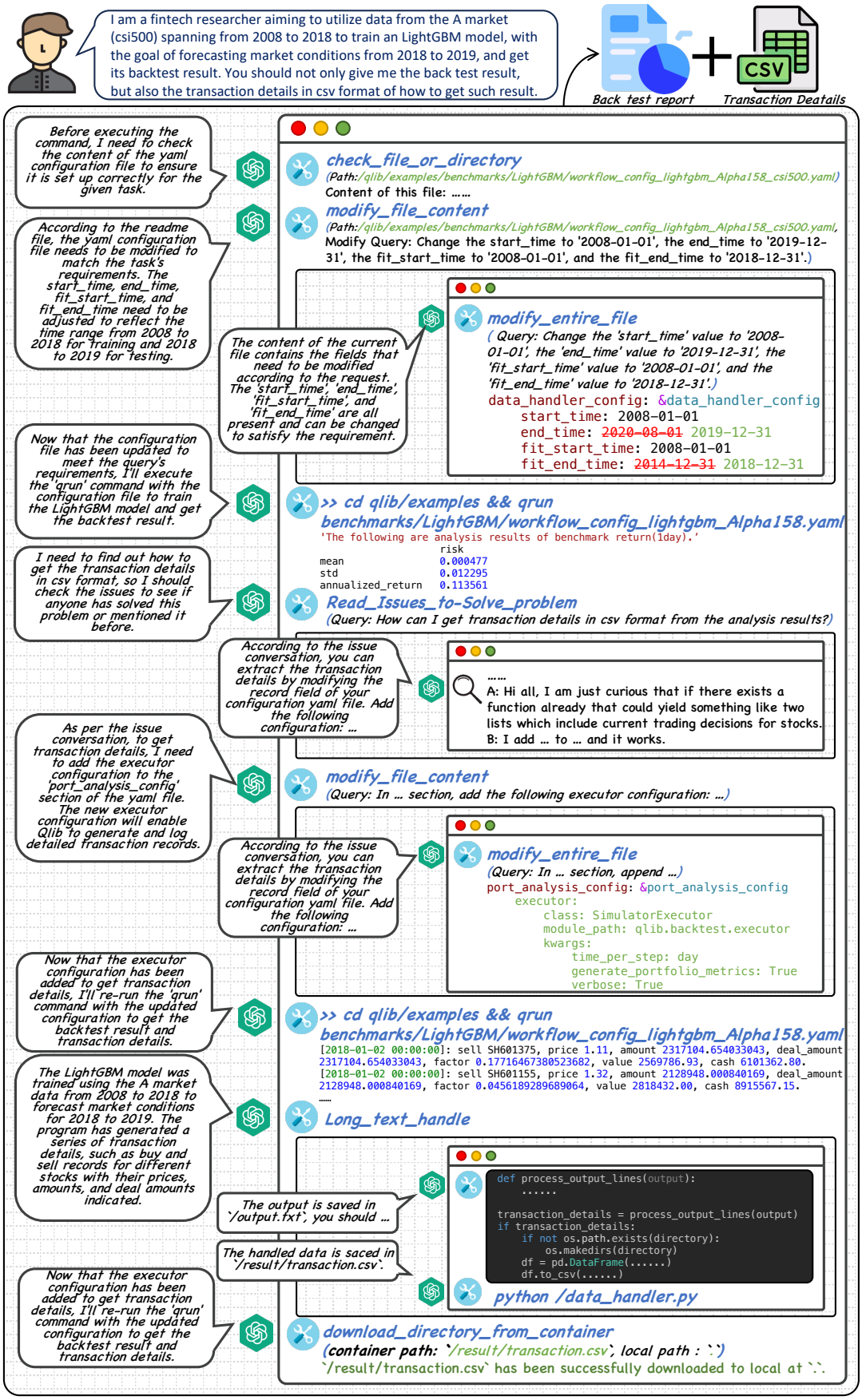

Figure 9: An example about the repository apply of `Qlib`.