GitTaskBench: A Benchmark for Code Agents Solving Real-World Tasks Through Code Repository Leveraging

Ziyi Ni[1,2*], Huacan Wang[3*†], Shuo Zhang[4*], Shuo Lu[1*], Ziyang He[5*], WangYou[6], Zhenheng Tang[7], Sen Hu[8], Bo Li[7], Chen Hu[6†], Binxing Jiao[6], Daxin Jiang[6], Yuntao Du[9,10†], Pin Lyu[1†]

[1]Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Science
[3]Midea Group
[4]School of Artificial Intelligence, Beijing University of Posts and Telecommunication
[5]National University of Singapore
[6]StepFun
[7]CSE, The Hong Kong University of Science and Technology
[8]Peking University
[9]C-FAIR&School of Software, Shandong University
[10]State Key Lab. for Novel Software Technology, Nanjing University
wanghc141@midea.com, hatcher@stepfun.com, yuntaodu@sdu.edu.cn, pin.lv@ia.ac.cn

Abstract

Beyond scratch coding, exploiting large-scale code repositories (e.g., GitHub) for practical tasks is vital in real-world software development, yet current benchmarks rarely evaluate code agents in such authentic, workflow-driven scenarios. To bridge this gap, we introduce GitTaskBench, a benchmark designed to systematically assess this capability via 54 realistic tasks across 7 modalities and 7 domains. Each task pairs a relevant repository with an automated, human-curated evaluation harness specifying practical success criteria. Beyond measuring execution and task success, we also propose the alpha-value metric to quantify the economic benefit of agent performance, which integrates task success rates, token cost, and average developer salaries. Experiments across three state-of-the-art agent frameworks with multiple advanced LLMs show that leveraging code repositories for complex task solving remains challenging: even the best-performing system, OpenHands+Claude 3.7, solves only 48.15 % of tasks. Error analysis attributes over half of failures to seemingly mundane yet critical steps like environment setup and dependency resolution, highlighting the need for more robust workflow management and increased timeout preparedness. By releasing GitTaskBench, we aim to drive progress and attention toward repository-aware code reasoning, execution, and deployment—moving agents closer to solving complex, end-to-end real-world tasks.

Code & Datasets — https://github.com/QuantaAlpha/GitTaskBench
Extended version — https://arxiv.org/abs/2508.18993

[*]Core contributions.
[†]Corresponding authors.

Introduction

In just two years, fueled by the transformative progress of large language model (LLM) agents, an increasing number of code benchmarks have reached saturation (Chen et al. 2021; Austin et al. 2021a; Hendrycks et al. 2021; Lu et al. 2021). However, **most early code-agent works and benchmarks target isolated, static problems**, such as algorithmic tests (Hendrycks et al. 2021; Li et al. 2022; Zheng et al. 2025), code completion at the function (Chen et al. 2021; Austin et al. 2021a; Lai et al. 2023), class (Du et al. 2023), or repository level (Liu, Xu, and McAuley 2023; Ding et al. 2023), program synthesis (Austin et al. 2021b), and program repair (Jimenez et al. 2023)—**failing to assess agents' capacity in real-world problem solving**.

Recent efforts have begun to develop more practical, comprehensive benchmarks. Some works still focus on code generation, requiring agents to produce increasingly complex code, even generating entire repositories from scratch (Yu et al. 2024; Ihle 2025; Chan et al. 2025; Miserendino et al. 2025; Starace et al. 2025). However, such a heavy burden remains prohibitively difficult for most current agent systems (Li et al. 2024; Starace et al. 2025). Moreover, **focusing solely on code generation overlooks the broader scope of real-world developer practice** (Masood 2024), and **provides diminishing insight into agent capabilities** (Gao et al. 2024). Another line of work rethinks evaluation paradigms (Ishibashi and Nishimura 2024) by integrating code generation with external tools or API calls (Li et al. 2023; Wang et al. 2024; Ye et al. 2024; Zhuo et al. 2024; Tang et al. 2025; Dong et al. 2025), thus easing the generation burden but still sidestepping the harder challenge of understanding and repurposing the full repositories.

However, *real-world programmers usually exploit open-source libraries to tackle diverse real-world tasks with-*

out reinventing "the wheel". Current GitHub has 28 million repositories and 190 million public projects to be exploited. Previous code-agent benchmarks ignore the ability of **autonomous environment setup and leveraging open-source repositories for solving complex, end-to-end tasks**, which is a more ***user-centric*** setting in practical software engineering (Lyu et al. 2023; Tang et al. 2023; Wang et al. 2025a).

To this end, we design and develop **GitTaskBench** (GitTaskBench 2025), which **systematically evaluates how well agents leverage code repositories to automatically solve real-world tasks end-to-end in realistic scenarios**, focusing on the following three key dimensions:

- Overall coding mastery: Navigating extensive documentation, understanding code dependencies, and dynamically generating, modifying, or debugging code.
- Task-oriented execution: Efficiently comprehending user intent, completing tasks via multi-turn reasoning and appropriate tool usage. All generated code is task-focused.
- Autonomous environment provisioning: Independently managing environment setup and dependency resolution in the sandbox without pre-built support.

The construction of GitTaskBench follows a rigorous four-step process: task and repository selection, completeness verification, execution framework design, and evaluation framework development, each performed by humans and some assisted by LLMs. The resulting benchmark covers 54 real-life, multimodal tasks across 7 domains and 24 subdomains, *going far beyond the technically narrow scope of traditional machine learning tasks* (Liu et al. 2018; Tang et al. 2023; Chan et al. 2025). Each task comes with human-designed, automated evaluation scripts that assess both execution completion and task pass by practical success criteria.

Beyond these core metrics, we further introduce the **alpha metric, which jointly considers cost and effectiveness**. Previous work has rarely analyzed or quantified the tangible benefits of agent applications, especially in multimodal scenarios (Yang et al. 2024; Maslej et al. 2025; Chen et al. 2025). Our alpha metric integrates task completion quality, agent token usage, and market-rate human labor costs into a unified framework, enabling direct, interpretable comparisons between agent and human efficiency.

Experiments are conducted on multiple code agents with advanced LLMs, and the results show the following findings: (1) Complex repository-centric tasks remain challenging, with the top success rate of only 48.15% (OpenHands, Claude3.7). (2) Replacing humans with agents is not always cost-effective; evaluating cost-efficiency is key for practical application. (3) Agents excel in purely textual tasks versus multimodal ones. (4) Better environment configuration and dependency management in the experimental workflow are crucial for accelerating real-world code agent deployment.

Our main contributions are summarized as follows:

1. We present GitTaskBench, *the first open-source benchmark* that tests agents on **solving real-world complex tasks by leveraging open-source repositories in a human-like manner**, encompassing 54 tasks drawn from 18 GitHub projects across 7 modalities.
2. **Each task includes hand-crafted test scripts and corresponding practical success criteria** *to enable rigorous and automated evaluation*.
3. We propose a novel domain-specific "alpha value" formula to quantitatively assess agent economic benefits, providing actionable insights for agent deployment.
4. We benchmark state-of-the-art agent frameworks with both open- and closed-source LLMs, perform hyperparameter sensitivity analysis, and conduct a detailed error analysis to highlight the remaining challenges.

Related Work

Existing code-agent benchmarks can broadly be divided into two categories: code-generation- and task-solving-centric.

In the first category, benchmarks evaluate code generation tasks of increasing complexity and granularity (Chen et al. 2021; Austin et al. 2021a; Du et al. 2023; Austin et al. 2021b; Hendrycks et al. 2021; Li et al. 2022; Liu, Xu, and McAuley 2023; Ding et al. 2023). More recently, more challenging benchmarks like SWE-Bench (Jimenez et al. 2023) have targeted resolving repo-level issues, SWE-Lancer (Miserendino et al. 2025) expands into real-world software engineering jobs with payouts, but most tasks remain narrowly bug fixing in pre-configured environments. These benchmarks share two main limitations: (1) tasks are still relatively isolated with small granularity, and (2) evaluations typically occur within simplified or synthetic environments rather than dynamic, realistic conditions.

In the second category, task-oriented benchmarks evaluate general programming skills involving tool usage and external API calls (Wang et al. 2022; Jain et al. 2022; Zhang et al. 2023; Lai et al. 2023; Wang et al. 2024; Ni et al. 2024; Tang et al. 2023; Chan et al. 2025; Wang et al. 2025b). However, these tasks remain predominantly technical-oriented, missing a critical capability **widely practiced**: leveraging GitHub repositories to **solve real-world daily problems**.

GitTaskBench

GitTaskBench rigorously evaluates code agents on realistic, repository-centric tasks closely aligned with common user queries (see Figure 1). Agents must autonomously analyze and reuse existing repositories to complete tasks that mirror authentic user workflows, handling any errors *without human intervention*. The benchmark is primarily handcrafted and validated by five computer science PhDs to ensure quality. Each task pairs a representative full-scale GitHub repository accompanied by a specific natural-language instruction specifying input-output requirements, and tailored *task-specific* evaluation metrics reflecting both correctness and utility, allowing meaningful automated assessment of agent performance. Below is how we constructed it.

Task and Repository Selection

We began by **identifying the target domains** through extensive literature reviews, deep LLM-driven research, and consultation with domain experts, combining these insights with practical, everyday experience. For each domain, we

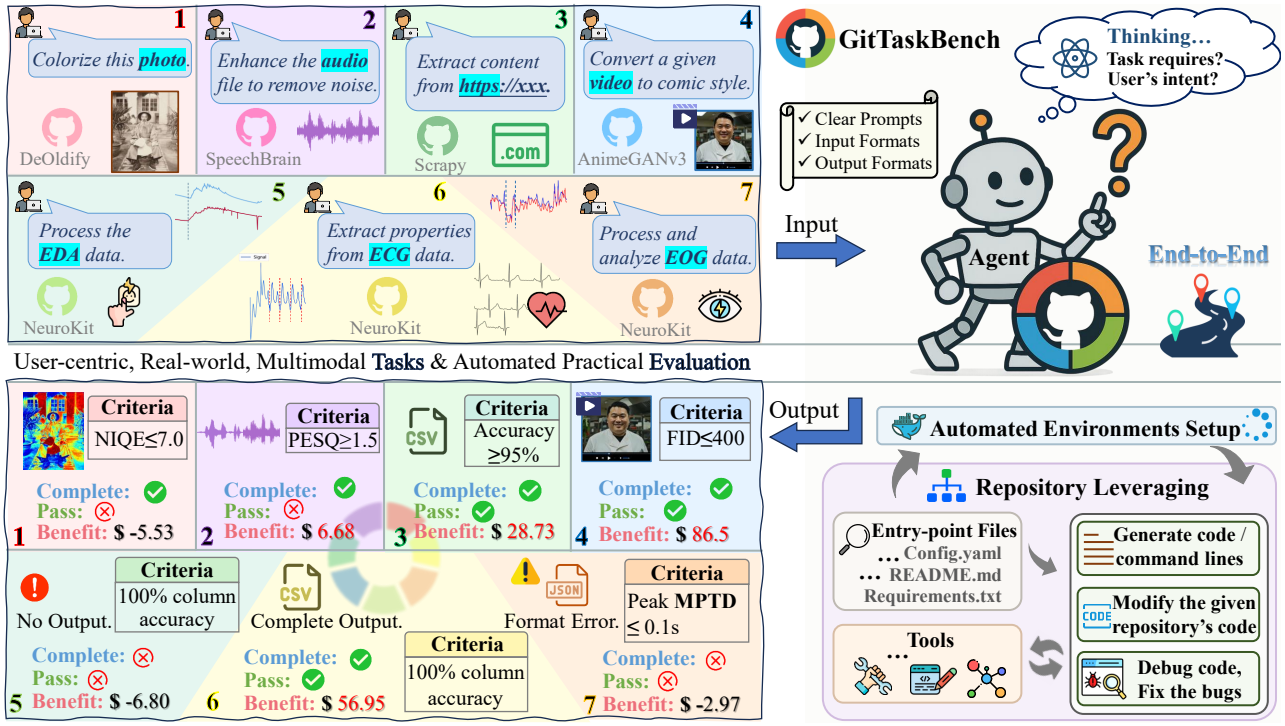

Figure 1: Overview of GitTaskBench. 7 example real-life tasks from different modalities and their evaluations are shown.

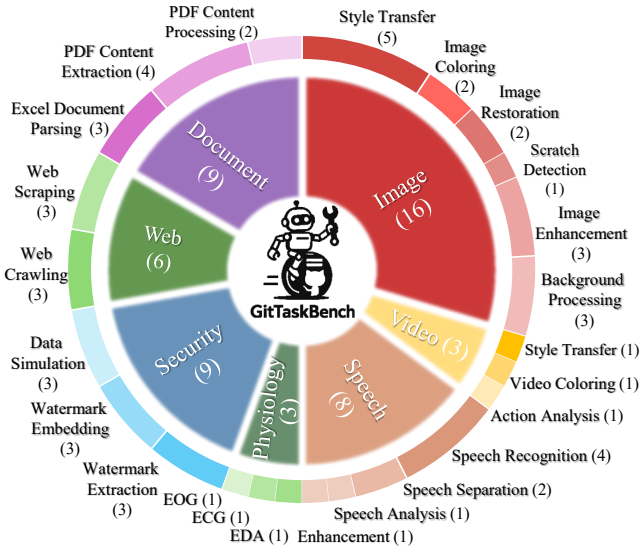

Figure 2: Overview of Task Domains in GitTaskBench.

selected subdomains that mirror frequent user needs, including a broad spectrum of modalities. We prioritized *tasks that are non-trivial, typically requiring the integration or reuse of existing tools or codebases*, to ensure that benchmarks are both meaningful and challenging for code agents. **Human completion time** for these tasks ranged up to three hours, averaging **1.34 hours** per task.

For each domain, we run targeted deep researches to **locate suitable GitHub repos**. Candidates must (1) be Python-based, (2) have $\geq$ 50 stars with activity in the past five years (including issue updates), and (3) provide ready-to-use weights and a simple setup. We then inspect key statistics like stars, forks, license, commit history, and manually verify functionality. The resulting set formed the pool of potential repositories.

Task and repository selection was iterative and tightly coupled—repository capabilities and task requirements were refined in parallel, with each informing the other. When a promising repository was identified, we would **expand potential task formulations** around its core features.

Summary statistics are presented in Table 2. Figure 2 illustrates the features of each domain. GitTaskBench supports both data generation and analysis-oriented objectives. See the Appendix in the extended version for details.

Completeness Verification

Following the **Repository Selection** phase, each chosen repository undergoes a stringent **Completeness Verification**. In this human-driven stage, experts follow the repository's documented instructions, performing tasks exactly as an agent would, ensuring both **a 100% human success rate and outputs that satisfy all task requirements**. This process confirms that the repository is fully operational and free of hidden obstacles that could hinder execution.

The verification process includes: Checking for essential *dependencies*, such as `requirements.txt` or `package.json`; Confirming the availability of key *configuration* files, like `config.yaml` or `setup.py`; Ensuring that the *required datasets* and *pre-trained models* are publicly accessible and properly formatted.

Benchmark	Task Num	Task Type	Multimodal	Repo Use	Repo-level CodeGen	Auto Env Setup
RepoBench (Liu, Xu, and McAuley 2023)	7778	Code Completion		✓		
Swe-Bench-Verified (Jimenez et al. 2023)	500	Program Repair		✓		
LiveCode (Jain et al. 2024)	584	Programming Competitions				
MLAgentBench (Huang et al. 2023)	13	ML ***Tasks***	✓		✓	
MLE-Bench (Chan et al. 2025)	72	Kaggle (ML) ***Tasks***	✓		✓	
PaperBench (Starace et al. 2025)	20	Paper Code Replication ***Tasks***	✓		✓	✓
GitTaskBench (Ours)	**54**	**User-centric, Daily-life *Tasks***	✓	✓	✓	✓

Table 1: Comparison of GitTaskBench (Ours) with Existing Benchmarks of Similar Complexity and Comprehensiveness.

Category	Metric	(Mean) Value
Instances	# Domain	7
	# Subdomain	24
	# Tasks	54
	# Modality	7
Repos	# Size	18
	# Files	204 (7–1157)
	# Classes	263.61 (2–1130)
	# Functions	1274.78 (25–4915)
	# Dependency	1242.72 (33–6979)
	# Calls	8651.28 (180–40552)
	# Code Lines	52.63 (0.575–351.42) **k**
	# Tokens	448.95 (4.87–2888.35) **k**

Table 2: Summary Statistics of GitTaskBench.

If any required resources are gated or instructions are only available via external links, we supplement the repository by downloading relevant files and inlining essential documentation into the `README.md`, ensuring all information needed for task execution is completely **self-contained**.

Execution Framework Design

To evaluate code agents in realistic and repository-leveraging contexts, we design an execution framework that integrates structured task formulation, automated execution, and output verification. This framework not only tests agents' capabilities for understanding and utilizing existing codebases but also ensures reproducibility and automation throughout the evaluation process.

Task Formulation. We meticulously define each task, specifying the expected input format (e.g., image path, text string) and the desired output format (e.g., processed image, generated report), ensuring clarity in task goals and reducing ambiguity in prompt interpretation. A single repository may host multiple distinct tasks, each with its clear definition.

Agent Inputs and Expected Outputs. The framework provides the agent with two inputs: a GitHub repository and a task definition prompt. Unlike conventional code generation settings that require only code snippets, our framework emphasizes end-to-end functionality. Agents are expected to return the final task-specific output, which could be a file, text, or visual result, depending on the task requirements.

Execution Workflow. Agents are evaluated on their ability to autonomously solve tasks in the multi-stage process: *(1) Repository Understanding:* Agents not just read the repository's code, but analyze its structure, dependencies, and available functionalities, often leveraging entry-point documentation such as `README.md` and selectively parsing key source files. *(2) Code Generation or Modification:* Based on their understanding and the task definition, agents generate new scripts or adapt existing files to fulfill the task. *(3) Environment Setup:* Agents are expected to construct the required execution environment, including issuing installation commands (e.g., `pip install -r requirements.txt`) and resolving dependency issues. *(4) Code Execution:* The generated or modified code is executed automatically in the sandbox, directly assessing the agent's ability to produce runnable and correct solutions.

Evaluation Framework

To support automated, practical, and cost-benefit evaluation, we introduce the following metrics, which are implemented through hand-verified custom-built test scripts.

Execution Completion Rate (ECR). ECR measures the proportion of cases where the agent successfully executes the target code repository and generates outputs in an acceptable format (e.g., `.jpg` or `.png` for image processing tasks). This metric reflects the agent's compatibility with the code repository and its basic operational capability. It ensures that: (1) the output file(s) exist, (2) the output file(s) are not empty, and (3) the output format can be correctly processed by the testing scripts.

Task Pass Rate (TPR). TPR quantifies the agent's actual performance quality in task completion. It is determined by formulating evaluation test functions and defining concrete success and failure criteria using established metrics tailored to each task, drawing on standards recognized within the domain developer community. TPR requires the agent's outputs to satisfy predefined quality standards, such as functional correctness, result completeness, or achieving specific task objectives. For example, in speech enhancement tasks, success might be defined by achieving a **PESQ** ≥ 2.0 (indicating acceptable perceptual quality) and a **SNR** $\geq 15dB$ (suggesting good suppression of noise). Tasks failing to meet these thresholds are marked as failures.

Both of the above metrics are evaluated using hand-crafted test scripts. Additionally, we streamlined the benchmarking process so that all tasks can be automatically assessed with a single shell command. The evaluation outputs a clear "Process" and "Result" status (success or failure), along with detailed "Comments" explaining the outcome—such as which metric exceeded a threshold, which criterion caused failure, or any error messages encountered during execution. See Appendix A for example test results.

Alpha Practical Value Assessment. We introduce a new perspective for evaluating LLM agents by incorporating market-driven cost considerations. While technical met-

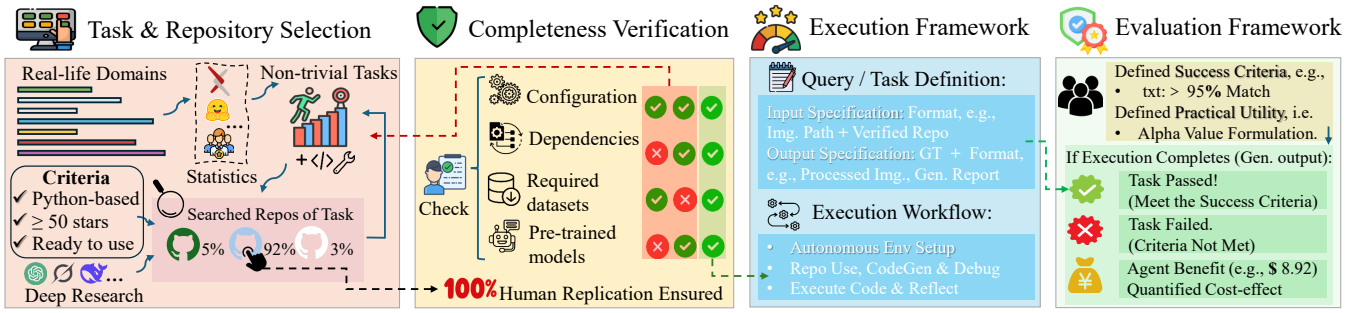

Figure 3: Overview of the GitTaskBench Data Curation and Processing Pipeline.

rics like ECR and TPR are essential, they overlook cost-effectiveness. High technical performance alone does not guarantee practical utility—an agent is only valuable if it completes tasks more cheaply than human labor, without sacrificing quality. In practice, agents incur tangible operational costs, like API fees for proprietary LLMs or hardware expenses for open-source solutions. We estimate the economic value of agent-completed tasks by quantifying potential cost savings, efficiency gains, and market impact from automation and scalability of these tasks. Accordingly, we propose the **α-score**, a value-based metric defined as the average net benefit generated by the agent across tasks:

$$\alpha = \frac{1}{n}\sum_{i=1}^{n}\left[(T \times MV \times Q) - C\right] \quad (1)$$

where n is the number of tasks in the evaluated area; T is a binary indicator of task success (1 if the agent successfully executes the target code repository, 0 otherwise), consistent with the definition of ECR; MV represents the estimated, prevailing market value of the task if completed by a human; Q is a quality factor (ranging from 0 to 1) that measures how closely the agent's output approximates the groundtruth produced by a human executing the same code repository; and C denotes the agent's total operational cost, which is approximated here as the API cost. The resulting **α-score** clearly reflects the economic viability and gains of the agent-based automation approach across the evaluated areas.

Human Check. Experts compare the automatically generated assessment of a repository/task with their own manual execution and evaluation, identifying any discrepancies or inconsistencies. For generating the groundtruth, humans can interpret the task requirements and iteratively adjust repository parameters to obtain the best possible output.

Because this expert-guided result provides a reliable upper bound on quality, we derive Q through human assessment. Five raters independently compare each agent output with the groundtruth and assign it to one of five levels—far below human (`0`), large gap (`0.25`), moderate gap (`0.50`), near parity (`0.75`), or indistinguishable from/better than human (`1`). The level chosen by the majority is recorded as the final Q value. MV is drawn from publicly listed freelance fees on the platforms (Upwork 2025; Fiverr 2025; Freelancer 2025) for similar deliverables—for example, roughly $10 per restored photo on Fiverr—providing a consistent task-level benchmark for the α-score.

Experiments

Setup

We evaluate three representative open-source frameworks, **Aider** (Aider-AI 2025), **OpenHands** (Wang et al. 2025c), and **SWE-Agent** (Yang et al. 2024) with multiple advanced models, including the closed-source as well as the open-source. For robustness, all reported results are averaged over two independent runs under identical settings.

Comparative Analysis

Different framework–LLM pairings exhibit substantial performance disparities, affecting both effectiveness (ECR, TPR) and efficiency (token usage, cost, API calls).

OpenHands achieves the best overall performance across all frameworks. As shown in Table 3, **(1)** OpenHands+Claude 3.7 delivers the best results (ECR 72.22%, TPR 48.15%) among all evaluated settings. **(2)** With the same LLM, OpenHands consistently outperforms Aider and SWE-Agent, likely due to its robust code execution capabilities and more proactive and explorative strategies.

OpenHands offers higher success rates, while SWE-Agent balances moderate cost and efficiency as a lower-cost alternative. SWE-Agent consistently uses fewer tokens than OpenHands when paired with top-performing closed-source models, indicating stronger control over context token usage. Meanwhile, Aider+DeepSeek V3 yields the lowest cost ($< \$0.003$) with reasonable output.

GPT-4.1 is more cost-efficient than Claude. (1) In SWE-Agent, Claude 3.7 leads but costs 2x more than 2nd-place GPT-4.1. **(2)** Under OpenHands, GPT-4.1 also delivers the 2nd-best ECR/TPR at just 1/10 or 1/30 of Claude's cost.

Open-source models generally underperform closed ones. But Qwen3-32B (with think mode) is impressive—reaching up to 60% of top closed Claude3.5's performance with far lower token usage. In contrast, Gemini 2.5 Pro underwhelms in think mode, likely due to the added context burden in our long, token-heavy, complex real-world tasks.

These findings highlight *trade-offs between performance, cost, and interaction complexity* when choosing agents—framework and model combinations. Next, we drill down into domain-specific performance, as visualized in Figure 4.

Agents perform notably better on purely textual tasks compared to multimodal, model-based tasks. Specifically, **(1)** most agents process office documents effectively, such as parsing Excel files with `Eparse` or splitting PDFs

Framework	LLM	ECR (%) ↑	TPR (%) ↑	Input Tokens (k) ↓	Output Tokens ↓	Cost ($) ↓
Aider	GPT-4o	5.56	1.85	10.67	**492.67**	0.0316
	GPT-4.1	11.11	7.41	14.83	734.17	0.0355
	Claude 3.5	16.67	12.96	**7.48**	534.00	0.0304
	DeepSeekV3	**20.37**	**16.67**	7.51	599.64	**0.00269**
SWE-Agent	GPT-4o	17.58	10.19	275.53	1282.70	0.778
	GPT-4.1	38.89	31.48	301.11	2098.33	0.661
	o3-mini	25.93	20.37	158.45	**215.20**	0.175
	Claude 3.5	41.67	22.23	455.34	943.30	1.38
	Claude 3.7	**64.81**	**42.59**	552.79	807.63	1.67
	DeepSeekV3	18.52	12.04	412.65	1649.82	**0.113**
	Qwen3-32b*	7.41	3.70	1445.97	2405.00	-
	Qwen3-32b*†	16.67	11.11	**124.15**	559.11	-
	Llama3.3-70b*	25.83	18.52	397.03	1985.64	-
OpenHands	GPT-4o	21.30	14.82	760.53	3990.31	1.94
	GPT-4.1	55.56	42.59	465.94	1535.47	**0.942**
	o3-mini	29.63	22.22	2523.53	183637.53	3.58
	Claude 3.5	53.70	40.74	2858.00	24929.47	8.95
	Claude 3.7	**72.22**	**48.15**	9501.25	85033.05	29.8
	Gemini-2.5-pro	51.85	35.19	760.88	35173.29	2.18
	DeepSeekV3	45.37	26.85	4717.78	31957.67	1.31
	Qwen3-8b*	1.85	1.85	846.26	2045.00	-
	Qwen3-14b*	11.11	5.56	339.42	2540.17	-
	Qwen3-32b*	35.19	25.93	591.02	2097.89	-
	Qwen3-32b*†	44.44	29.63	208.00	8755.35	-
	Llama3.3-70b*	27.78	20.37	**132.69**	**872.93**	-

Table 3: Performance Comparison of Different Frameworks and LLMs on GitTaskBench. Bold values indicate the best among all models for each metric; underlined values denote the second-best. The best-performing row for each metric is highlighted. All token values are rounded to two decimal places. * means our self-deployed model. †: with think mode.

using `PyPDF`. That is because these workflows typically require reading simple wrapper scripts that import the library API. **(2)** In contrast, multimodal tasks, especially in image or speech processing domains, mainly involve model-based processing and prediction and thus demand much deeper competence. For example, removing image scratches with `DeScratch` entails installing multiple dependencies, downloading pretrained weights, and configuring runtime arguments—all of which require a nuanced understanding of the repository's build and execution process.

Current agents often struggle with such complex workflows, suggesting that **future work should focus on richer codebase comprehension and automated environment management** beyond what a simple `README` scan provides.

Sensitivity Analysis to Configuration Changes

Since OpenHands consistently outperformed SWE-Agent overall, we examined how its key hyperparameters affect performance. Notably, GPT-4o with OpenHands lagged behind despite strong overall results, with execution traces revealing *frequent failures from environment setup errors and flawed code generation*. To clarify these issues, we tested two critical hyperparameters. `timeout`: Maximum time per iteration. `max_iteration`: Total environment interactions allowed. Performance variation is in Table 5.

Results show that more generous settings significantly boost performance. Increasing the `timeout` (from 120s to 1800s) raises both ECR and TPR, but also incurs more tokens, indicating that **environment setup may be the primary time-consuming step in repurposing repositories**. Similarly, increasing `max_iteration` (from 30 to 100) consistently improves ECR and TPR, suggesting that **more interaction rounds could help mitigate errors within reasonable limits**. Overall, these findings underscore the importance of tuning both interaction depth and time budgets to balance effectiveness and computational efficiency.

Practical Benefits Analysis

Given OpenHands' strong overall performance, we select it as the evaluation backbone for estimating the practical value of the three most cost-effective models. We treat each repository as a domain, capturing agents' domain-specific applicability and their ability to leverage the code. For each repository, we computed α-score by averaging net gains of all n tasks executed with that repository, as defined in Eq (1).

Figure 6 (a) presents the α score (green revenue minus red cost) of each repository, facilitating a direct cost-benefit comparison across models and repositories. Figure 6 (b) displays the Pareto curves, illustrating how each model's total alpha is distributed across repositories. The dashed 45-degree reference line represents a perfectly even distribution: if the cumulative alpha curve rises steeply and surpasses the diagonal early, it means just a few repositories contribute most of the total benefit (high concentration). In contrast, if a curve close to the diagonal means alpha is more evenly spread across repositories. This directly reveals the difference in benefit concentration for each model.

Expensive tasks are always profitable if completed by the agent, while cheap tasks require careful cost control. Repositories with intrinsically high human market value (MV), like `VideoPose3D`, `FunASR`, and `NeuroKit`, yield the largest positive α when agents succeed. Low-MV

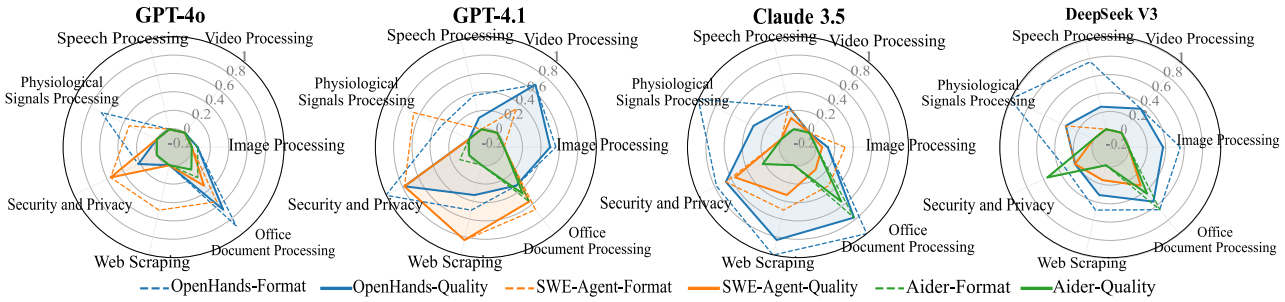

Figure 4: Performance Evaluation of GPT-4o, GPT-4.1, Claude 3.5, DeepSeek V3 across Different Task Domains.

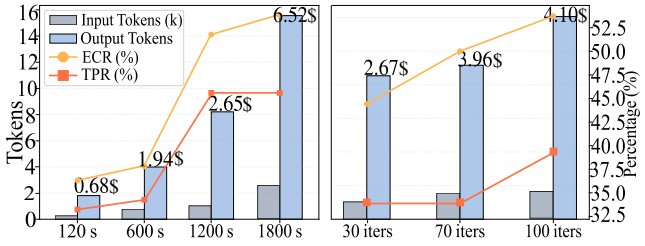

Figure 5: Effect of Timeout (max_iteration = default) and Max Iteration (timeout = 600s) on OpenHands (GPT-4o).

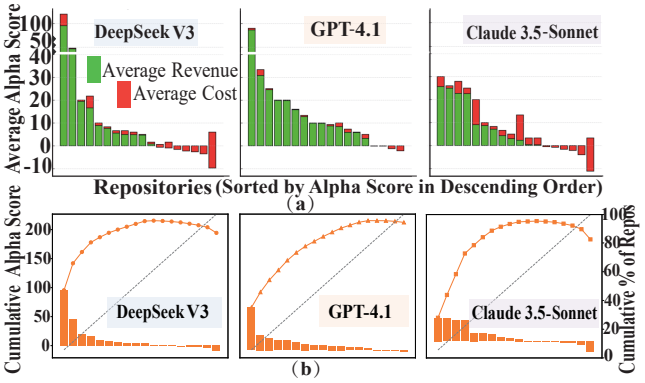

Figure 6: The α per Repository (a) and Pareto Curves (b).

image-processing tasks ($MV \approx$ \$5–10) often produce negative α once the agent's average cost exceeds \$1–2. This pattern underscores the importance of controlling operational costs for tasks with limited commercial potential.

DeepSeek V3 delivers the highest overall benefit and best cost–performance for most repositories. GPT-4.1's performance is more consistent and robust across scenarios, with fewer large losses. Claude 3.5 has the most dispersed returns, excelling at information extraction but being cost-sensitive on compute-intensive vision tasks.

α captures meaningful distinctions that technical metrics (ECR/TPR) alone may miss, emphasizing the need to align agent deployment with task-specific economic profiles.

Error Analysis

To better understand the challenges in such repository-centric tasks, we studied execution errors encountered across various agents. We grouped all errors into five types: **E1,** Environment-Setup; **E2,** Workflow Planning; **E3,** Repository Comprehension; **E4:** Runtime; and **E5:** Failures to Follow Instructions. Case studies are in extended version.

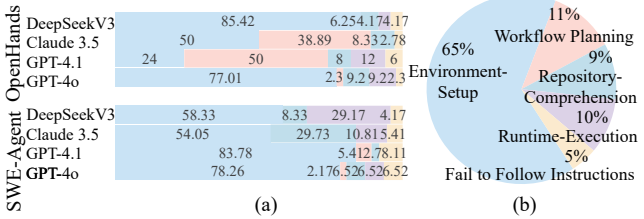

Figure 7: Distribution of Errors per Agent (a) and Overall Error Statistics (b).

As summarized in Figure 7 (b), E1 errors were the most common, constituting 65.04% of all failures. These usually came from dependency conflicts, missing binary wheels, or absent system-level libraries. Notably, env setup doesn't improve results but causes most failures—showing its unavoidable importance in real-world agent applications. E2 errors mainly reflected agents' inability to orchestrate execution sequences or their stagnation at setup stages. E3 errors occurred when agents misidentified entry-point scripts or misused APIs within repositories. E4 errors involved premature termination due to system freezes, timeouts, or interrupts (`Ctrl+C`). E5 errors, the least frequent, included things like wrong file naming, incorrect output formats, missing deliverables, or solving the task without using the repository as required. Check Appendix F for examples of each.

We compared errors across agents/models and found similar weaknesses regardless of architecture (see Figure 7 (a)). Key improvements could be summarized as: robust dependency management, enhanced execution planning, deeper repo comprehension, smarter resource handling during runtime, and rigorous instruction following—all crucial for more reliable and effective real-world agent performance.

Conclusion

We introduced GitTaskBench, a benchmark for evaluating agents' ability to leverage repository code for complex task solving. With carefully curated tasks, thoroughly reviewed repositories, and practical automated evaluation, GitTaskBench sets a new standard for assessing real-world agent utility. We hope this benchmark drives greater focus on repository utilization for everyday, non-technical challenges and encourages economic value-driven agent applications.

Acknowledgments

The research was supported by the National Science and Technology Major Project (2022ZD0116406), the project ZR2025QC1570 of the Shandong Provincial Natural Science Foundation, in part by an NSFC grant 62432008, RGC RIF grant R6021-20, an RGC TRS grant T43-513/23N-2, RGC CRF grants C7004-22G, C1029-22G and C6015-23G, NSFC/RGC grant CRS_HKUST601/24 and RGC GRF grants 16207922, 16207423 and 16203824.

References

Aider-AI. 2025. Aider: AI Pair Programming in Your Terminal. https://aider.chat/. Accessed: 2025-07-31.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021a. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021b. Program synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*.

Chan, J. S.; Chowdhury, N.; Jaffe, O.; Aung, J.; Sherburn, D.; Mays, E.; Starace, G.; Liu, K.; Maksin, L.; Patwardhan, T.; Weng, L.; and Mądry, A. 2025. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. arXiv:2410.07095.

Chen, K.; Ren, Y.; Liu, Y.; Hu, X.; Tian, H.; Xie, T.; Liu, F.; Zhang, H.; Liu, H.; Gong, Y.; et al. 2025. xbench: Tracking Agents Productivity Scaling with Profession-Aligned Real-World Evaluations. *arXiv preprint arXiv:2506.13651*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating Large Language Models trained on Code. *arXiv preprint arXiv:2107.03374*.

Ding, Y.; Wang, Z.; Ahmad, W.; Ding, H.; Tan, M.; Jain, N.; Ramanathan, M. K.; Nallapati, R.; Bhatia, P.; Roth, D.; et al. 2023. Crosscodeeval: A Diverse and Multilingual Benchmark for Cross-file Code Completion. *Advances in Neural Information Processing Systems*, 36: 46701–46723.

Dong, P.; Tang, Z.; Liu, X.; Li, L.; Chu, X.; and Li, B. 2025. Can Compressed LLMs Truly Act? An Empirical Evaluation of Agentic Capabilities in LLM Compression. In *Proceedings of the 42th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR.

Du, X.; Liu, M.; Wang, K.; Wang, H.; Liu, J.; Chen, Y.; Feng, J.; Sha, C.; Peng, X.; and Lou, Y. 2023. Classeval: A Manually-crafted Benchmark for Evaluating LLMs on Class-level Code Generation. *arXiv preprint arXiv:2308.01861*.

Fiverr. 2025. Fiverr Freelance Services. https://www.fiverr.com. Accessed: 2025-07-31.

Freelancer. 2025. Freelancer Marketplace. https://www.freelancer.com. Accessed: 2025-07-31.

Gao, C.; Lan, X.; Li, N.; Yuan, Y.; Ding, J.; Zhou, Z.; Xu, F.; and Li, Y. 2024. Large language Models Empowered Agent-based Modeling and Simulation: A Survey and Perspectives. *Humanities and Social Sciences Communications*, 11(1): 1–24.

GitTaskBench. 2025. GitTaskBench: Anonymous GitHub Repository. https://anonymous.4open.science/r/GitTaskBench-EE47/. Accessed: 2025-07-31.

Hendrycks, D.; Basart, S.; Kadavath, S.; Mazeika, M.; Arora, A.; Guo, E.; Burns, C.; Puranik, S.; He, H.; Song, D.; et al. 2021. Measuring Coding Challenge Competence with Apps. *arXiv preprint arXiv:2105.09938*.

Huang, Q.; Vora, J.; Liang, P.; and Leskovec, J. 2023. MLAgentBench: Evaluating language Agents on Machine Learning Experimentation. *arXiv preprint arXiv:2310.03302*.

Ihle, H. T. 2025. WeirdML Benchmark. https://htihle.github.io/weirdml.html.

Ishibashi, Y.; and Nishimura, Y. 2024. Self-organized agents: A LLM Multi-agent Framework toward Ultra Large-scale Code Generation and Optimization. *arXiv preprint arXiv:2404.02183*.

Jain, N.; Han, K.; Gu, A.; Li, W.-D.; Yan, F.; Zhang, T.; Wang, S.; Solar-Lezama, A.; Sen, K.; and Stoica, I. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Jain, N.; Vaidyanath, S.; Iyer, A.; Natarajan, N.; Parthasarathy, S.; Rajamani, S.; and Sharma, R. 2022. Jigsaw: Large Language Models Meet Program Synthesis. In *Proceedings of the 44th International Conference on Software Engineering*, 1219–1231.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. 2023. Swe-bench: Can Language Models Resolve Real-World GitHub Issues?? *arXiv preprint arXiv:2310.06770*.

Lai, Y.; Li, C.; Wang, Y.; Zhang, T.; Zhong, R.; Zettlemoyer, L.; Yih, W.-t.; Fried, D.; Wang, S.; and Yu, T. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, 18319–18345. PMLR.

Li, B.; Wu, W.; Tang, Z.; Shi, L.; Yang, J.; Li, J.; Yao, S.; Qian, C.; Hui, B.; Zhang, Q.; et al. 2024. Devbench: A Comprehensive Benchmark for Software Development. *CoRR*.

Li, M.; Zhao, Y.; Yu, B.; Song, F.; Li, H.; Yu, H.; Li, Z.; Huang, F.; and Li, Y. 2023. Api-bank: A Comprehensive Benchmark for Tool-augmented LLMs. *arXiv preprint arXiv:2304.08244*.

Li, Y.; Choi, D.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Dal Lago, A.; et al. 2022. Competition-level Code Generation with Alphacode. *Science*, 378(6624): 1092–1097.

Liu, T.; Xu, C.; and McAuley, J. 2023. Repobench: Benchmarking Repository-level Code Auto-completion Systems. *arXiv preprint arXiv:2306.03091*.

Liu, Y.; Zhang, H.; Zeng, L.; Wu, W.; and Zhang, C. 2018. MLench: benchmarking Machine Learning Services

Against Human Experts. *Proceedings of the VLDB Endowment*, 11(10): 1220–1232.

Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C.; Drain, D.; Jiang, D.; Tang, D.; et al. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Lyu, B.; Cong, X.; Yu, H.; Yang, P.; Qin, Y.; Ye, Y.; Lu, Y.; Zhang, Z.; Yan, Y.; Lin, Y.; et al. 2023. Gitagent: Facilitating Autonomous Agent with Github by Tool Extension. *arXiv preprint arXiv:2312.17294*.

Maslej, N.; Fattorini, L.; Perrault, R.; Gil, Y.; Parli, V.; Kariuki, N.; Capstick, E.; Reuel, A.; Brynjolfsson, E.; Etchemendy, J.; et al. 2025. Artificial intelligence index report 2025. *arXiv preprint arXiv:2504.07139*.

Masood, A. 2024. The Agentic Imperative Series Part 5: Manus and AutoGen. https://medium.com/p/41724fe77a77. Accessed: 2025-07-31.

Miserendino, S.; Wang, M.; Patwardhan, T.; and Heidecke, J. 2025. SWE-Lancer: Can Frontier LLMs Earn $1 Million from Real-World Freelance Software Engineering? *arXiv preprint arXiv:2502.12115*.

Ni, Z.; Li, Y.; Yang, N.; Shen, D.; Lv, P.; and Dong, D. 2024. Tree-of-Code: A Tree-Structured Exploring Framework for End-to-End Code Generation and Execution in Complex Task Handling. *arXiv preprint arXiv:2412.15305*.

Starace, G.; Jaffe, O.; Sherburn, D.; Aung, J.; Chan, J. S.; Maksin, L.; Dias, R.; Mays, E.; Kinsella, B.; Thompson, W.; et al. 2025. PaperBench: Evaluating AI's Ability to Replicate AI Research. *arXiv preprint arXiv:2504.01848*.

Tang, X.; Liu, Y.; Cai, Z.; Shao, Y.; Lu, J.; Zhang, Y.; Deng, Z.; Hu, H.; An, K.; Huang, R.; et al. 2023. ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code. *arXiv preprint arXiv:2311.09835*.

Tang, Z.; Liu, X.; Wang, Q.; Dong, P.; He, B.; Chu, X.; and Li, B. 2025. The Lottery LLM Hypothesis, Rethinking What Abilities Should LLM Compression Preserve? In *The Fourth Blogpost Track at ICLR 2025*.

Upwork. 2025. Upwork Online Freelance Marketplace. https://www.upwork.com. Accessed: 2025-07-31.

Wang, H.; Ni, Z.; Zhang, S.; Lu, S.; Hu, S.; He, Z.; Hu, C.; Lin, J.; Guo, Y.; Du, Y.; et al. 2025a. RepoMaster: Autonomous Exploration and Understanding of GitHub Repositories for Complex Task Solving. *arXiv preprint arXiv:2505.21577*.

Wang, Q.; Wang, T.; Tang, Z.; Li, Q.; Chen, N.; Liang, J.; and He, B. 2025b. MegaAgent: A Large-Scale Autonomous LLM-based Multi-Agent System Without Predefined SOPs. In *The 63rd Annual Meeting of the Association for Computational Linguistics*.

Wang, X.; Chen, Y.; Yuan, L.; Zhang, Y.; Li, Y.; Peng, H.; and Ji, H. 2024. Executable Code Actions Elicit Better LLM Agents. In *Forty-first International Conference on Machine Learning*.

Wang, X.; Li, B.; Song, Y.; Xu, F. F.; Tang, X.; Zhuge, M.; Pan, J.; Song, Y.; Li, B.; Singh, J.; Tran, H. H.; Li, F.; Ma, R.; Zheng, M.; Qian, B.; Shao, Y.; Muennighoff, N.; Zhang, Y.; Hui, B.; Lin, J.; Brennan, R.; Peng, H.; Ji, H.; and Neubig, G. 2025c. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. arXiv:2407.16741.

Wang, Z.; Zhou, S.; Fried, D.; and Neubig, G. 2022. Execution-based Evaluation for Open-domain Code Generation. *arXiv preprint arXiv:2212.10481*.

Yang, J.; Jimenez, C.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; and Press, O. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37: 50528–50652.

Ye, J.; Li, G.; Gao, S.; Huang, C.; Wu, Y.; Li, S.; Fan, X.; Dou, S.; Zhang, Q.; Gui, T.; et al. 2024. Tooleyes: Fine-grained Evaluation for Tool Learning Capabilities of Large Language Models in Real-world Scenarios. *arXiv preprint arXiv:2401.00741*.

Yu, Z.; Zhao, Y.; Cohan, A.; and Zhang, X.-P. 2024. HumanEval Pro and MBPP Pro: Evaluating Large Language Models on Self-invoking Code Generation. *arXiv preprint arXiv:2412.21199*.

Zhang, K.; Zhang, H.; Li, G.; Li, J.; Li, Z.; and Jin, Z. 2023. Toolcoder: Teach Code Generation Models to Use Api Search Tools. *arXiv preprint arXiv:2305.04032*.

Zheng, Z.; Cheng, Z.; Shen, Z.; Zhou, S.; Liu, K.; He, H.; Li, D.; Wei, S.; Hao, H.; Yao, J.; et al. 2025. LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming? *arXiv preprint arXiv:2506.11928*.

Zhuo, T. Y.; Vu, M. C.; Chim, J.; Hu, H.; Yu, W.; Widyasari, R.; Yusuf, I. N. B.; Zhan, H.; He, J.; Paul, I.; et al. 2024. Bigcodebench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. *arXiv preprint arXiv:2406.15877*.