



RepoMaster: Autonomous Exploration and Understanding of GitHub Repositories for Complex Task Solving

Huacan Wang^{1,*‡} Ziyi Ni^{1,2 *} Shuo Zhang^{3 *} Shuo Lu^{1,2} Sen Hu⁴

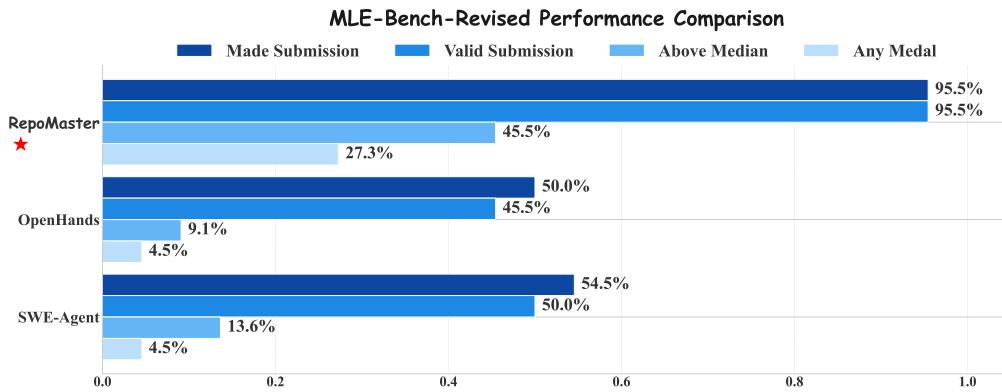
Ziyang He⁵ Chen Hu⁶ Jiaye Lin⁷ Yifu Guo⁸ Ronghao Chen⁴

Xin Li⁹ Dixin Jiang⁶ Yuntao Du^{10,‡} Pin Lyu^{2,‡}

¹UCAS ²CASIA ³BUPT ⁴PKU ⁵NUS ⁶StepFun ⁷THU ⁸SCNU ⁹USTC ¹⁰SDU

* These authors contributed equally to this work.

† Corresponding authors: wanghuacan17@mails.ucas.ac.cn, yuntaodu@sdu.edu.cn, pin.lv@ia.ac.cn



Abstract

The ultimate goal of code agents is to solve complex tasks autonomously. Although large language models (LLMs) have made substantial progress in code generation, real-world tasks typically demand full-fledged code repositories rather than simple scripts. Building such repositories from scratch remains a major challenge. Fortunately, GitHub hosts a vast, evolving collection of open-source repositories, which developers frequently reuse as modular components for complex tasks. Yet, existing frameworks like OpenHands and SWE-Agent still struggle to effectively leverage these valuable resources. Relying solely on README files provides insufficient guidance, and deeper exploration reveals two core obstacles: overwhelming information and tangled dependencies of repositories, both constrained by the limited context windows of current LLMs. To tackle these issues, we propose RepoMaster, an autonomous agent framework designed to explore and reuse GitHub repositories for solving complex tasks. For efficient understanding, RepoMaster constructs function-call graphs, module-dependency graphs, and hierarchical code trees to identify essential components, providing only identified core elements to the LLMs rather than the entire repository. During autonomous execution, it progressively explores related components using our exploration tools and prunes information to optimize context usage. Evaluated on the adjusted MLE-bench, RepoMaster achieves a 110% relative boost in valid submissions over the strongest baseline OpenHands. On our newly released GitTaskBench, RepoMaster lifts the task-pass rate from 40.7% to 62.9% while reducing token usage by 95%. Our code and demonstration materials are publicly available at <https://github.com/QuantaAlpha/RepoMaster>.

1 Introduction

In recent years, the integration of toolchains [1, 2, 3, 4] and iterative reasoning [5, 6, 7, 8] has significantly enhanced large language models (LLMs) in code-related tasks [9, 10, 11]. These advancements have enabled LLMs to proficiently complete code snippets [12, 13], debug errors [14], and even address complex machine learning problems [15, 16]. However, when confronted with real-world challenges that necessitate task-driven code repositories [17], they struggle. At present, tackling such tasks remains largely manual and time-consuming due to the complexity and scale of the required code, which makes purely generative approaches impractical [11, 18, 19]. To overcome this, we propose a paradigm shift: reuse and adapt existing repositories as modular components tailored to specific tasks. This approach not only mitigates the challenges associated with repository-level code generation but also supports the broader goal of enabling agents to autonomously address sophisticated tasks using simple natural language instructions [20, 21].

To facilitate this approach, leveraging platforms like GitHub becomes crucial. With over 28 million public repositories out of 190 million total projects, GitHub offers an extensive library of ready-made solutions for code agents [17, 18, 22, 23]. Developers frequently reuse these repositories to tackle complex problems, yet LLM-based systems still falter in fully automating this process. Although frameworks like OpenHands [24] and SWE-Agent [14] demonstrate strong general capabilities, they often stumble on real-world codebases. In practice, simply following README instructions seldom works: READMEs can be vague, incomplete, or even erroneous, and repositories are not guaranteed to match a task’s requirements out of the box—commands may need parameter changes, and key files can be misplaced. Consequently, when agents fail to locate or execute the necessary code, they must adapt by modifying existing components or generating new code to bridge the gap.

To achieve it, agents need to understand the repository in a task-driven way. However, GitHub repositories often have two key properties that make this hard: (1) intricate structural complexity, with many interconnected files, classes, and functions, and (2) information density that exceeds the context limits of most LLMs. Existing frameworks [14, 15, 24, 25] do not provide mechanisms for grasping repository structures, tracking detailed dependencies, or strategically managing information within these constraints, ultimately resulting in suboptimal performance and higher token cost.

In this paper, we introduce RepoMaster, an end-to-end agent framework designed for automating the use of code repositories to tackle complex tasks. To address these challenges, RepoMaster draws inspiration from human programmers, who rarely read every line of code or error log when exploring unfamiliar codebases. Instead, they first map a project’s structure, start viewing a key file, then jump to its relevant files based on signals like error traces, and filter out irrelevant details.

Following this intuition, RepoMaster first performs hierarchical structure analysis, builds dependency and call graphs, and identifies core components as the initial context. Navigated by these connections, it progressively explores the repository and applies information selection when viewing files and execution feedback to keep each interaction concise. By iteratively applying these steps, RepoMaster mimics human prioritization and makes efficient use of limited context windows. When evaluated on both MLE-R—a revised version of MLE-Bench-Lite [16]—and our newly constructed GitTaskBench [26], RepoMaster achieves significantly higher completion and success rates than OpenHands and SWE-Agent, while using far fewer tokens. Our contributions are summarized as follows:

(1) We propose a novel automated framework, **RepoMaster**, that can effectively leverage code repositories to solve the complex real-world tasks end-to-end. (2) To efficiently comprehend code in a goal-oriented, human-like manner, we integrate hybrid structural hierarchy modeling with core component identification, context-aware code exploration, and efficient information selection. (3) We validate RepoMaster’s effectiveness and efficiency against Openhands and SWE-agent through experiments on diverse complex tasks from the MLE-R and GitTaskBench.

2 Related Work

2.1 Code Generation

LLMs have made substantial progress in code generation [12, 13, 27, 28], exemplified by closed-source models [29, 30, 31] and the open-source series [32, 33, 34, 35]. Beyond basic code completion [36], modern LLMs now support advanced tasks such as semantic code editing [23, 37],

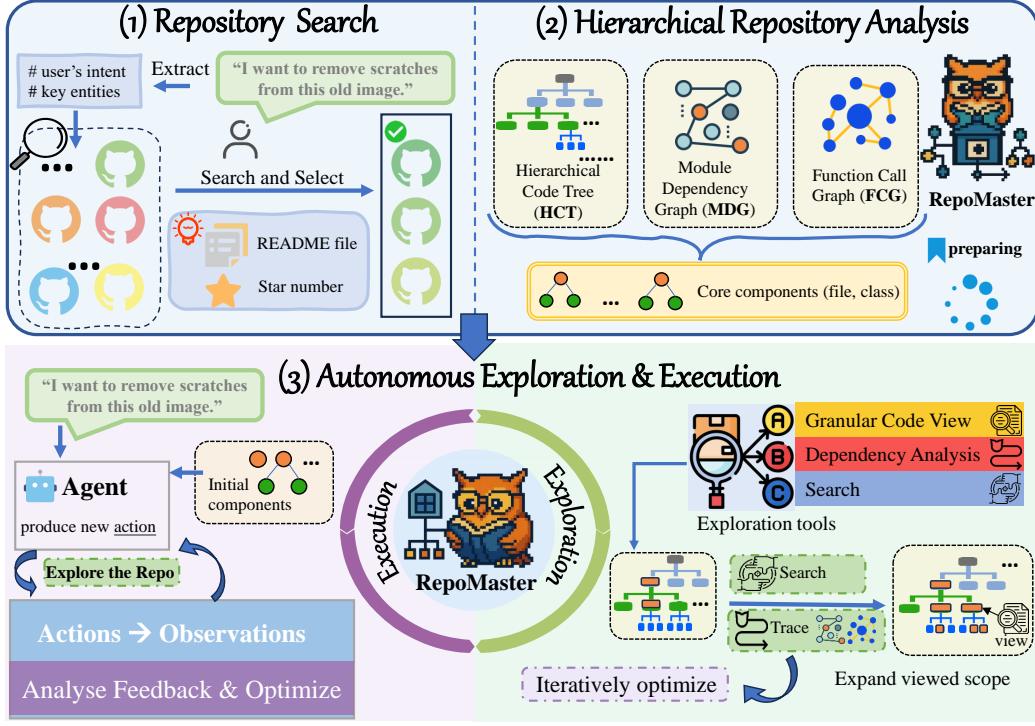


Figure 1: Overview of RepoMaster, consisting of Repository Search, Hierarchical Repository Analysis and Autonomous Exploration & Execution.

debugging [38], and generating machine learning pipelines (e.g., AIDE [25] and MLAB [15] for Kaggle competitions). However, fully automating the creation of complex real-world codebases from scratch remains a critical challenge for AI agents [16, 19, 22].

2.2 LLM-based Agents for Tool Use

External tools are essential for extending the capabilities of LLM agents [5, 39, 40]. Relying on executable code [6, 9]—using scripts to import inherent libraries, or call APIs, functionalized tools—has become a mainstream paradigm. Current works mainly focus on “tool learning” [1, 6, 10], but the more essential aspect of where to find the right tools is relatively overlooked [41]. Benchmarks, such as API-Bank [42] and ToolEyes [43], synthesize function libraries but are not realistic or practical; platforms such as RapidAPI [44] host real services but are closed-source and hard to extend. Standards such as FastAPI [45] or MCP [46], which unify interfaces for tool use via function calling mechanisms, have emerged. However, GitHub—a rich and dynamic ecosystem for automatically creating tools—remains underutilized in this context. Although GitAgent [17] first explored GitHub repositories as a tool extension, it is limited by simplistic repository search and understanding, and lacks validation in diverse real-world scenarios.

2.3 Repository Utilization

Using GitHub repositories to solve complex real-world tasks presents significant challenges. RepoAgent [47] produces high-level documentation but fails to include realistic, task-oriented usage examples. ML-Bench-A [18] focuses on setting up the environment rather than understanding the repository. OpenHands [24] and SWE-Agent [14] are strong general agents that use step-by-step prompting to break down tasks and write code, but they lack methods to deeply understand the repository structure or build a clear hierarchy of its components. Aider [48] can track file dependencies but misses detailed function-level connections and cannot autonomously explore the codebase. Interactive assistants like Copilot [49] and Cursor [50] are effective for small-to-medium projects but struggle in large-scale repository contexts due to limited dependency awareness.

3 Method

Most current frameworks follow the CodeAct paradigm [9, 14, 21, 24], offering basic file-editing and exploration commands (e.g., OpenHands’ AgentSkills [24] and SWE-Agent’s command set [14]). But relying on README-based mappings and simple find/edit operations misses many core components and cannot perform deeper, autonomous exploration within limited LLM contexts. In contrast, RepoMaster mimics human programmers by performing a static, structure-aware analysis to locate critical components, then dynamically selecting only the essential snippets—skipping irrelevant information and focusing the LLM’s limited context on what matters. The full end-to-end RepoMaster framework consists of three stages: (1) Repository Search: Identifying repositories relevant to the task. (2) Hierarchical Repository Analysis: Preparing the structures for exploration. (3) Autonomous Exploration & Execution: Iteratively interact with the repository and adjust exploration actions based on execution feedback. An overview of the framework is provided in Figure 1.

3.1 Repository Search

To address complex online tasks expressed in natural language, we develop a deep-search method to locate the GitHub repositories most relevant to the task. We begin by analyzing the user’s intent and extracting key entities to target the suitable repositories. We examine their README file and star count to assess their relevance and potential, and provide a brief description. Then, we select them by content quality and practical utility. Finally, we validate the top three candidates and deliver the results as structured JSON. An example of the deep-searching log is shown in Appendix B.

3.2 Hierarchical Repository Analysis

3.2.1 Hybrid Structural Repository Mapping

An essential prerequisite for task-oriented repository automation is a comprehensive structural model of the codebase. We sanitize the repository by removing all non-source files, retaining only executable .py files. For each retained file, we perform a single Abstract Syntax Tree (AST) walk [51] to recursively harvest both the *meta-information* and the *raw source snippet* of every module, class, and function. These atomic units provide the basis for understanding the repository’s structure.

Let the target repository be denoted $\mathcal{R} = \langle M, C, F, \mathcal{I} \rangle$, where $M = \{m_1, \dots, m_{|M|}\}$ is the set of modules (one per .py file), $C = \{c_1, \dots, c_{|C|}\}$ the set of classes, $F = \{f_1, \dots, f_{|F|}\}$ the set of functions/methods, and $\mathcal{I} \subseteq M \times M$ the explicit import relations captured from source files. On this foundation, we construct three complementary artefacts:

- **Hierarchical Code Tree (HCT).** \mathcal{T} , a nested package → module → class → function containment map annotated with line counts and docstring snippets.
- **Function Call Graph (FCG).** $G_f = (V_f = F, E_f, w_f)$, where an edge $(f_i, f_j) \in E_f$ exists if f_i invokes f_j ; the weight w_f encodes call frequency.
- **Module Dependency Graph (MDG).** $G_m = (V_m = M, E_m, w_m)$, in which $(m_i, m_j) \in E_m$ if m_i explicitly depends on m_j ; w_m measures coupling strength.

We thus obtain the tuple $\langle M, C, F, \mathcal{I}, G_f, G_m, \mathcal{T} \rangle$, providing the agent with a deterministic, loss-minimal structural synopsis of the entire repository before any task-specific exploration.

3.2.2 Core Component Identification

Having obtained a fine-grained yet verbose structural synopsis of the repository, we now need to compress this information into a concise context that preserves only the most influential code entities—small enough for multiple interaction turns within the LLM’s window, yet rich enough to preserve global semantics. To this end, we specify an **importance scoring scheme** that operates first at the module level and then propagates to classes.

Module-level scoring. Each module $m \in M$ receives a score $I(m) \in [0, 10]$ by linearly aggregating six orthogonal features,

$$s = [\text{Dependency}, \text{Complexity}, \text{Usage}, \text{Semantic}, \text{Doc}, \text{Git}], \quad (1)$$

$$I(m) = \min\left(\sum_{i=1}^6 w_i s_i(m), 10\right), \quad w_i \equiv 1, \quad (2)$$

where *Dependency* captures centrality in MDG using the personalized PageRank [52] algorithm, *Complexity* approximates cyclomatic complexity, *Usage* measures import and call frequency, *Semantic* flags high-value keywords (e.g., *main*, *core*), *Doc* quantifies docstring richness, and *Git* reflects commit volume and recency. Detailed formulas for each feature are deferred to Appendix F.

Class-level refinement. Module scores serve as priors for class importance. For every class c located in module $\mu(c)$, we compute

$$J(c) = I(\mu(c)) + \frac{|F_c|}{\max_{c'} |F_{c'}|} + \frac{\text{Calls}(F_c)}{\max_{c'} \text{Calls}(F_{c'})}, \quad (3)$$

where F_c denotes the method set of class c . The second term rewards class richness in functionality; the third term captures how often its methods are actually invoked in the repository. Classes are ranked according to $J(c)$, and the top- k classes are selected as the repository's *core components*.

3.2.3 Repository Context Initialization

Building on the identified core components, we construct an initial repository context in four distinct blocks. First, we include the complete README.md file, which provides high-level descriptions and detailed usage guidance authored by human developers. Second, we append a series of concise natural-language summaries for the highest-priority modules, giving the LLM a brief overview of each critical script's purpose. Third, we provide the source code of core components (i.e., the classes scored and selected in Section 3.2.2) as fine-grained semantic anchors. Finally, for all other top-ranked modules, we provide a flat, directory-grouped list of their file paths for easy on-demand lookup. Figure 2 illustrates the initial context, and Appendix D provides a complete example of this initial repository context construction.

This structured context serves as the agent's "launchpad" for dynamic exploration, allowing it to prioritize high-impact modules, trace dependencies, formulate targeted code queries and select relevant classes or functions, bridging static analysis with dynamic reasoning and task execution.

3.3 Autonomous Exploration & Execution

3.3.1 Context-aware Code Exploration

Once the agent has internalized the repository's functionality and overall structure, it immediately transitions to dynamic analysis, performing an autonomous, hierarchical and graph-based traversal of the codebase. To support in-depth comprehension and effective utilization of the repository, we offer a suite of fine-grained exploration tools organized into three categories: **Granular Code View**, **Dependency Analysis**, and **Search**.

- **Granular Code View.** This tool enables the agent to inspect the implementation details of files, classes, and functions using the HCT. It also retrieves and exposes the repository's directory hierarchy, facilitating swift orientation within the codebase.
- **Dependency Analysis.** This tool traces call chains and dependency paths by analyzing the FCG and MDG, respectively. It uncovers complex invocation and dependency relationships among code entities, thereby deepening the agent's comprehension of module interactions and overall code structure.
- **Search.** This tool equips the agent with robust search capabilities, facilitating rapid location of specific code segments within large and intricate codebases. It employs keyword matching to ensure efficient retrieval of relevant entities.

Together, these tools empower AI agents to proactively and autonomously navigate and examine code repositories, achieving a level of comprehension and flexibility comparable to human developers. Empirically, we observe that complex repositories typically require detailed dependency analysis using FCG and MDG, whereas simpler repositories often allow agents to effectively rely on HCT.

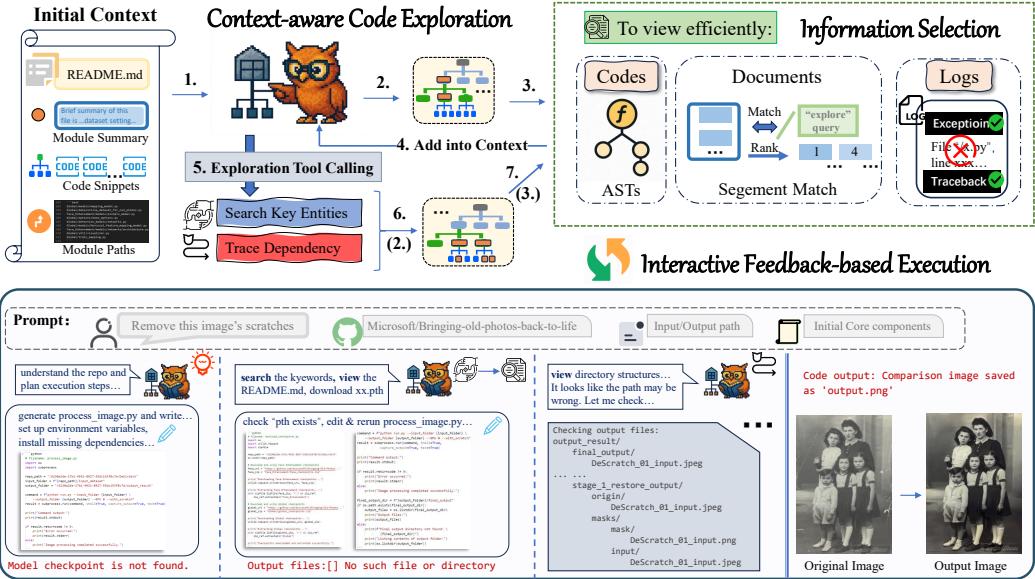


Figure 2: Overview of RepoMaster’s autonomous exploration–execution loop and an example demonstration. The agent begins by analyzing the initial context (Step 1) and specifies a file to inspect (Step 2). For efficient viewing, it extracts only the key information from that file (Step 3) and appends it to the context (Step 4). In the next exploration–execution iteration (Step 6→2, Step 7→3), the agent uses exploration tools to identify additional relevant files and repeats context-aware code exploration. Once it has gathered enough information, RepoMaster alternates between writing and running “.py” scripts, handling errors, and debugging based on feedback until the task is completed.

3.3.2 Interactive Feedback-based Execution

Task execution is grounded in the agent’s evolving understanding of the repository. Once the agent has identified the hybrid structural elements described in Section 3.2.1 and core components described in Section 3.2.2 relevant to a given task, it begins to perform task-oriented operations.

Crucially, execution and exploration form a continuous, interleaved loop rather than a linear sequence. The agent can fluidly alternate between writing code and locating files, viewing content and reading logs, or tracing dependencies, all driven by the task context across different interaction turns, and powered by the exploration tools described in Section 3.3.1. This flexible loop allows the agent to iteratively refine its behavior by retrieving just-in-time information from the codebase. Figure 2 illustrates the execution and exploration pipeline of RepoMaster.

3.3.3 Context-aware Information Selection For Efficient Viewing

The agent must juggle source code, documentation, execution results and logs within a tight LLM token window for multiple turns, making it difficult to maintain a globally coherent view of the repository and severely limiting its applicability to large projects. To mitigate this issue, we propose a multi-level content reduction strategy that retains only the most critical information.

Viewing Code. At the code level, the agent parses source files into Abstract Syntax Trees (ASTs), extracts semantically and structurally meaningful subtrees, and uses these extracted subtrees as inputs.

Viewing Documents. For large or unstructured artifacts (e.g., .txt or .csv files), the agent divides each file into fixed-length chunks of L_c tokens. It then generates retrieval prompts tailored to the current subtask, ranks the chunks by relevance, and retains the top n_c most relevant segments.

Viewing Feedback Logs. At the log level, we apply a human-like debugging heuristic that retains only the opening and closing segments of the log (where command invocations, exception traces, and diagnostic results cluster) and discards verbose intermediate output.

Multi-level reduction strategies activate only when the combined size of all candidate inputs exceeds the per-interaction token limit L , preserving global coherence by focusing on high-impact information and ensuring each execution-loop step relies on a compact, relevant context.

4 Experiments

4.1 Benchmarks and Metrics

To validate the effectiveness of RepoMaster, we evaluate it using two benchmarks.

MLE-R. The original MLE-Bench [16] derives from Kaggle competitions, designed to evaluate LLM agents’ capabilities in end-to-end machine learning engineering tasks. To construct MLE-R, we select 22 MLE-Bench tasks (covering nearly all MLE-Bench-lite cases) and apply the search procedure described in Section 3.1 to retrieve suitable GitHub repositories for each task, ensuring a fair comparison¹; the tasks’ requirements are set to be completed based on their chosen repository rather than generating code from scratch.

Performance in MLE-R is evaluated using a medal-based system, the same as the original MLE-Bench, where solutions are assessed based on official Kaggle thresholds² for gold, silver, and bronze medals. Metrics include the achieved score, medal thresholds, and medal qualification, providing a clear indication of the model’s proficiency in competitive ML engineering tasks.

GitTaskBench. In contrast to MLE-R, which emphasizes standard machine learning tasks (e.g., image classification), our new proposed GitTaskBench [26]³ benchmark evaluates LLM agents on more practical real-world problems—common tasks whose complexity or format largely demands leveraging existing repositories, such as photo restoration. The benchmark consists of 18 repositories and 54 tasks, all described in natural language and designed to be completed using the provided repositories across a wide range of domains, such as image processing, video analysis, speech, physiological signals, office automation, and security and privacy.

GitTaskBench evaluates two key aspects: **Execution Completion Rate** (measuring the model’s ability to leverage the repository for output) and **Task Pass Rate** (assessing whether the output meets task-specific evaluation criteria). Given the diversity of tasks, evaluation metrics are predefined and tailored within the benchmark, ensuring a comprehensive assessment. Note that total tokens include both input and output tokens.

4.2 Evaluation Setup

We evaluate our approach against two baseline frameworks and compare the performance across three state-of-the-art LLMs. The evaluation setup is as detailed below.

Baseline Frameworks.

We evaluate two baseline frameworks: OpenHands [24] and SWE-agent [14]. OpenHands provides sandboxed environments for code execution and API interactions, while SWE-agent focuses on autonomous GitHub issue resolution.

Large Language Models. We evaluate multiple leading LLMs, including the closed-source GPT-4o-2024-08-06 [53] and Claude-3-5-sonnet-20241022 [54], as well as the open-source DeepSeek V3-0324 [55]. This setup enables a comprehensive assessment of both agent architectures and LLM capabilities on solving real-world tasks with repository utilization.

Implementation Details. Our proposed solution RepoMaster is built on a multi-agent dialog platform AutoGen [21]. To ensure agent performance, we set a few key hyperparameters. Specifically, we set the maximum token length per interaction L to 8000 tokens. For initial context construction, we generate concise summaries for the top 20 modules by importance score and extract $k = 10$ key classes. During the feedback phase, unstructured text files are split into chunks of $L_c = 1000$ tokens, retaining the $n_c = 4$ most relevant segments.

¹Ensure a fair comparison, as other general agent frameworks do not support automatic repository retrieval.

²The specific thresholds for gold, silver, and bronze medals are provided in Appendix G.

³More detailed descriptions can be found in Appendix A.

Table 1: Performance comparison of different frameworks and LLMs on MLE-R. The best performance is bolded, and the second-best is underlined; the same is below.

Framework	LLM	Made Submission (%)	Valid Submission (%)	Above Median (%)	Bronze (%)	Silver (%)	Gold (%)	Any Medal (%)
SWE-Agent	GPT-4o	72.73	54.55	0.00	0.00	0.00	0.00	0.00
	Claude 3.5	54.55	50.00	13.64	0.00	0.00	4.55	4.55
	DeepSeek V3	54.55	36.36	4.55	0.00	0.00	4.55	4.55
OpenHands	GPT-4o	50.00	45.45	0.00	0.00	0.00	0.00	0.00
	Claude 3.5	50.00	45.45	9.09	0.00	0.00	4.55	4.55
	DeepSeek V3	63.64	36.36	0.00	0.00	0.00	0.00	0.00
RepoMaster	GPT-4o	<u>86.36</u>	<u>86.36</u>	<u>36.36</u>	4.55	0.00	13.64	18.18
	Claude 3.5	95.45	95.45	45.45	4.55	0.00	<u>22.73</u>	27.27
	DeepSeek V3	95.45	<u>86.36</u>	<u>36.36</u>	4.55	4.55	<u>13.64</u>	<u>22.73</u>

4.3 Comparison with SOTA

On the MLE-R benchmark, RepoMaster with Claude 3.5 attains a 95.45% valid submission rate and a 27.27% medal acquisition rate (including 22.73% gold medals), representing a more than five-fold improvement over the best open-source Agent baseline. RepoMaster with GPT-4o also achieves a strong 86.36% valid submission rate and 18.18% medal rate, further confirming its robust performance advantage under varied settings.

RepoMaster’s significant performance improvement stems primarily from its effective identification and utilization of core components within open-source repositories, such as neural network architecture designs, optimized hyperparameter configurations, and data preprocessing pipelines. In contrast, baseline methods like OpenHands and SWE-Agent often struggle to pinpoint critical modules during repository exploration, filling limited context windows with excessive irrelevant code, resulting in insufficient understanding of model architectures and training logic.

In the GitTaskBench evaluation, RepoMaster significantly outperforms existing open-source frameworks SWE-Agent and OpenHands. Based on Claude 3.5, RepoMaster achieves a 75.92% execution completion rate and 62.96% task pass rate, surpassing OpenHands (53.70%, 40.74%) and SWE-Agent (41.67%, 22.23%). Similarly, RepoMaster maintains significant advantages on GPT-4o and DeepSeek V3, demonstrating that RepoMaster’s inherent capabilities have good universality across underlying models. More importantly, RepoMaster substantially reduces computational overhead, with token consumption when using Claude 3.5 approximately 95% lower than OpenHands (154k vs 2883k tokens/task), proving the effectiveness of our hybrid hierarchical structure analysis and information pruning strategies.

Table 2: Performance comparison of different frameworks and LLMs on GitTaskBench.

Framework	LLM	Execution Completion Rate (%) ↑	Task Pass Rate (%) ↑	#Total Tokens ↓
SWE-Agent	GPT-4o	17.58	10.19	276k
	Claude 3.5	41.67	22.23	456k
	DeepSeek V3	18.52	12.04	414k
OpenHands	GPT-4o	21.30	14.82	765k
	Claude 3.5	53.70	40.74	2883k
	DeepSeek V3	45.37	26.85	4750k
RepoMaster	GPT-4o	48.14	40.74	<u>250k</u>
	Claude 3.5	75.92	62.96	<u>154k</u>
	DeepSeek V3	<u>61.11</u>	<u>44.44</u>	255k

4.4 Insightful analysis

Ablation Study To quantitatively assess the contribution of each component in RepoMaster, we conduct a comprehensive ablation study on the GitTaskBench benchmark using GPT-4o as the underlying model. By systematically removing key mechanisms, we measure their impact on three metrics of effectiveness and efficiency: execution completion rate, task pass rate, and token usage. The results are shown in Table 3.

Table 3: Ablation study on the impact of core mechanisms in RepoMaster with GPT-4o on the GitTaskBench Benchmark. The "w/o All" refers to RepoMaster without any of the three mechanisms. "Exec. Comp." stands for "Execution Completion."

Configuration	Exec. Comp. Rate (%)↑	Task Pass Rate (%)↑	#Total Tokens↓
RepoMaster (Full)	48.14	40.74	250k
- w/o Hybrid Hierarchical Analysis	46.29 (-1.85%)	38.88 (-1.86%)	274k
- w/o Code Exploration	44.44 (-3.7%)	35.18 (-5.56%)	284k
- w/o Information Selection	46.29 (-1.85%)	38.88 (-1.86%)	265k
- w/o All	29.62 (-18.52%)	24.07 (-16.67%)	112k

Hybrid Hierarchical Analysis: Removing this component causes slight decreases in execution completion and task pass rates, with other components partially compensating. Token usage increases by 9.67%, showing that without structural repository understanding, more exploration is required.

Context-aware Code Exploration: This component provides a suite of exploration tools enabling agents to autonomously navigate codebases according to task requirements, inspect file contents, and trace call chains and dependencies. The experiments prove this component to be particularly critical, with its removal causing significant decreases in execution and pass rates. More notably, token usage increases substantially, validating our hypothesis that strategic exploration effectively reduces repository exploration needs, allowing the agent to precisely target task-relevant components.

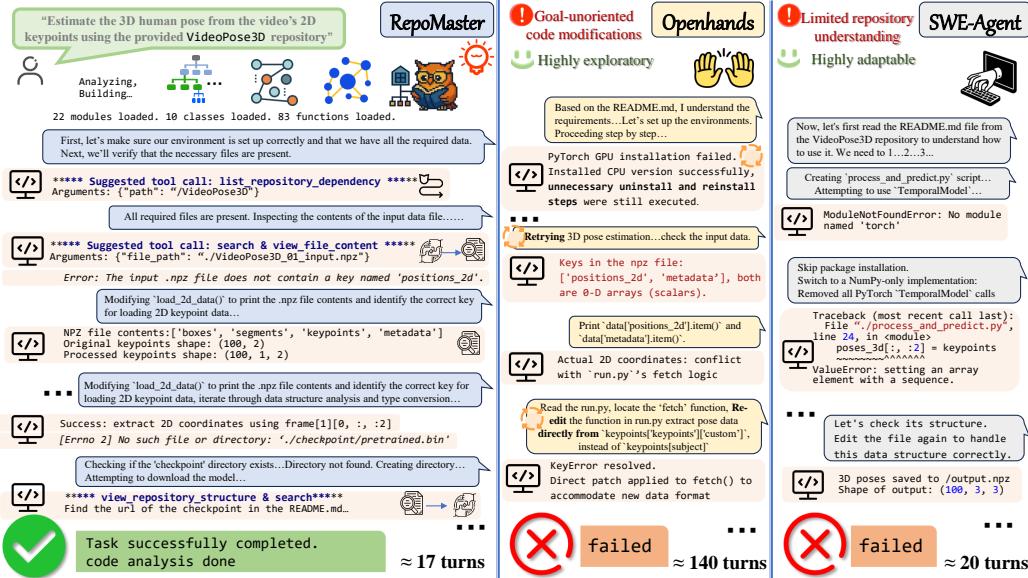


Figure 3: RepoMaster vs. OpenHands and SWE-Agent: A Comparative Case Study.

Information Selection: This component applies multi-level content reduction strategies to extract key information from source code, documents, and execution logs, preventing the context window from being filled with irrelevant content. Removing it decreases performance similarly to removing hybrid hierarchical analysis, with token usage increasing by only 6.00%. Its main value is maintaining a high signal-to-noise ratio rather than reducing token consumption.

The most revealing comparison is between the full RepoMaster system and the base code agent without any of our proposed components. The baseline achieves only **29.62%** execution completion and **24.07%** task pass rates—decreases of 18.52% and 16.67%. Interestingly, the baseline’s token usage is significantly lower, but this reflects a failure case rather than efficiency: the agent simply gives up earlier without the necessary tools to effectively explore and utilize the repository.

Further analysis of the failure modes in ablated systems reveals: Without hybrid hierarchical analysis, the agent struggles to locate key repository components, often getting lost in non-essential files; without context-aware exploration, the agent frequently explores irrelevant parts of the repository, resulting in context fragmentation and redundant exploration; without information selection, the agent’s context window becomes cluttered with low-value information, causing it to miss important details in error messages and execution traces.

4.5 Case Study

For the case study, we evaluated RepoMaster against OpenHands and SWE-Agent on a challenging 3D pose estimation task from GitTaskBench. As shown in Figure 3, neither baseline completed the task due to different failure modes. OpenHands ran extensive trial-and-error iterations (~ 140 attempts, $>10\times$ others) and consumed higher tokens without success. SWE-Agent, although quicker, lacked task-level repository understanding—treating each error as a standalone fix and defaulting to a coarse 3D pose method that strayed from the core algorithm, causing task degradation. In contrast, RepoMaster leveraged structured repository analysis to efficiently focus on key components, achieving successful task completion with fewer attempts (<20 iterations).

5 Conclusion

We introduce **RepoMaster**, an end-to-end autonomous agent framework designed for automating the use of code repositories to tackle complex tasks. By combining static structural analysis of the repository with autonomous exploration, RepoMaster outperforms OpenHands and SWE-Agent in two challenging benchmarks. These results demonstrate that treating open-source repositories as modular, composable *tools*—rather than burdens to be regenerated from scratch—forms a powerful paradigm for solving complex real-world tasks. Beyond performance gains, RepoMaster promotes a more sustainable and collaborative AI-for-code ecosystem. Its capacity to reuse and adapt existing repositories lays the groundwork for large-scale orchestration of multiple projects within a single workflow, automated propagation of bug fixes and security patches upstream, and straightforward transfer to domains that share analogous structural challenges, such as hardware description languages, robotic middleware, or data-centric notebook collections. By enabling agents to understand and integrate code in context, RepoMaster accelerates the virtuous cycle between human contributors and AI systems, fostering continual improvement across the open-source landscape.

References

- [1] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.
- [2] Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*, 2024.
- [3] Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024.
- [4] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.
- [5] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [6] Ziyi Ni, Yifan Li, Ning Yang, Dou Shen, Pin Lv, and Daxiang Dong. 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*, 2024.
- [7] Guangsheng Bao, Hongbo Zhang, Linyi Yang, Cunxiang Wang, and Yue Zhang. Llms with chain-of-thought are non-causal reasoners. *arXiv preprint arXiv:2402.16048*, 2024.
- [8] Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.
- [9] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.
- [10] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*, 2024.
- [11] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.
- [12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [13] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.
- [14] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.
- [15] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.
- [16] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.
- [17] Bohan Lyu, Xin Cong, Heyang Yu, Pan Yang, Yujia Qin, Yining Ye, Yaxi Lu, Zhong Zhang, Yukun Yan, Yankai Lin, et al. Gitagent: facilitating autonomous agent with github by tool extension. *arXiv preprint arXiv:2312.17294*, 2023.

- [18] Xiangru Tang, Yuliang Liu, Zefan Cai, Yanjun Shao, Junjie Lu, Yichi Zhang, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, et al. Mi-bench: Evaluating large language models and agents for machine learning tasks on repository-level code. *arXiv preprint arXiv:2311.09835*, 2023.
- [19] Bowen Li, Wenhan Wu, Ziwei Tang, Lin Shi, John Yang, Jinyang Li, Shunyu Yao, Chen Qian, Binyuan Hui, Qicheng Zhang, et al. Devbench: A comprehensive benchmark for software development. *CoRR*, 2024.
- [20] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.
- [21] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.
- [22] Xiao Yu, Lei Liu, Xing Hu, Jacky Wai Keung, Jin Liu, and Xin Xia. Where are large language models for code generation on github? *arXiv preprint arXiv:2406.19544*, 2024.
- [23] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
- [24] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2024.
- [25] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixin Xu, Ian Kaplan, Deniss Jancenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.
- [26] GitTaskBench: Anonymous github repository. <https://anonymous.4open.science/r/GitTaskBench-EE47/>. Accessed: May 2025.
- [27] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- [28] Raymond Li, Yushi Yamaguchi, Sinong Liu, Amir Gal, Matthew Ju, Farhad Mireshghallah, Andrew Wang, Jiayi Gao, Wanjun Wang, Chen Lei, and et al. Starcoder: may the source be with you! *CoRR*, abs/2305.06161, 2023.
- [29] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [30] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [31] Anthropic. Introducing claude. <https://www.anthropic.com/index/introducing-claude>, 2023. Accessed: 2023-10-20.
- [32] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [33] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- [34] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

- [35] Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.
- [36] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, 2023.
- [37] Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, et al. Can it edit? evaluating the ability of large language models to follow code editing instructions. *arXiv preprint arXiv:2312.12450*, 2023.
- [38] Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, et al. Debugbench: Evaluating debugging capability of large language models. *arXiv preprint arXiv:2401.04621*, 2024.
- [39] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024.
- [40] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36:45870–45894, 2023.
- [41] Xiao Yu, Lei Liu, Xing Hu, Jacky Wai Keung, Jin Liu, and Xin Xia. Where are large language models for code generation on github? *arXiv preprint arXiv:2406.19544*, 2024.
- [42] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.
- [43] Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. Tooleyes: fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*, 2024.
- [44] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.
- [45] José Haro Peralta. *Microservice APIs: Using Python, Flask, FastAPI, OpenAPI and More*. Simon and Schuster, 2023.
- [46] Model context protocol: Introduction. <https://modelcontextprotocol.io/introduction>. Accessed: May 2025.
- [47] Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, et al. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667*, 2024.
- [48] Aider: Ai pair programming in your terminal. <https://aider.chat/>. Accessed: May 2025.
- [49] Github copilot: Your ai pair programmer. <https://github.com/features/copilot>. Accessed: May 2025.
- [50] Cursor: The ai code editor. <https://www.cursor.com/en>. Accessed: May 2025.
- [51] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 783–794, 2019.
- [52] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford infolab, 1999.

- [53] OpenAI. Hello gpt-4o. <https://openai.com/index/hello-gpt-4o/>, 2024. Accessed: 2024-05-18.
- [54] AI Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1, 2024.
- [55] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Appendix

A. Description of GitTaskBench

Although existing benchmarks focus predominantly on software engineering tasks, **GitTaskBench** focuses on everyday user queries drawn from domains such as image enhancement, video analysis, speech processing, physiological signal analysis, office automation, and security/privacy, tasks that are difficult to complete without a ready-made codebase. For every query, the benchmark provides one specific open source repository and requires the agent to solve the task by reusing and adapting that code. Task queries are presented in natural language to better simulate genuine user requirements.

To ensure the benchmark reflects realistic and meaningful usage scenarios, each task is paired with a carefully selected open-source repository that is highly relevant and capable of supporting the task. This design simulates a common real-world situation: given a specific user need and an existing codebase, the challenge lies in effectively applying and adapting the provided code to fulfill the requirement. By anchoring each task to a concrete repository, GitTaskBench evaluates not only general problem-solving skills but also the practical ability to navigate, comprehend, and leverage real-world projects.

To guarantee data quality and comprehensiveness, five computer science students with relevant domain expertise manually reviewed and validated each repository included in GitTaskBench. Necessary downloadable resources were supplemented, and any critical information absent from the original repositories was carefully added to ensure dataset completeness and feasibility. This rigorous validation process ensures all tasks are clearly defined, practically feasible, and accurately reflective of real-world development scenarios.

For evaluation, we establish a standardized multi-stage scoring system to comprehensively assess the correctness and completeness of solutions generated by various frameworks and LLMs. Specifically, the scoring system first evaluates technical feasibility by verifying whether the solution can be successfully executed, ensuring the completion of essential GitHub operations. Subsequently, the solutions are assessed from the user perspective, examining the extent to which actual user requirements have been met. Through this careful and structured design, GitTaskBench provides the research community with a valuable resource for advancing the evaluation and development of intelligent agents in real-world scenarios.

Listing 1: Example case of photo restoration task.

```
# Query
Restore an old photo by removing scratches from it.

# Input
- Repository: DeScratch (located at /mnt/DeScratch_01/repo/DeScratch)
- Input file: /mnt/DeScratch_01/repo/DeScratch/input_dataset/DeScratch_01_input.jpeg
- Type: Image file (JPEG format)
- Description: An old photograph containing visible scratches

# Output Requirements
- Output directory: /mnt/DeScratch_01/repo/DeScratch/output_result
- Output filename convention:
  * Single file: output.{extension}
  * Multiple files: output_01.{extension}, output_02.{extension}, etc.
- Output type: Restored image file

# Completion Criteria
1. Code execution completes without errors
2. Output file(s) exist in the specified directory
3. Output file(s) follow the required naming convention
4. Output file(s) are readable image files

# Success Criteria
1. Scratches are effectively removed from the input image
2. Image quality is maintained or improved
3. Original image content is preserved
```

B. Repository Search Pipeline

This section presents the condensed logs generated during the repository retrieval process. Listing 2 document search queries, repository rankings, and selection rationales, thereby ensuring transparency and reproducibility in our repository selection methodology.

Listing 2: Search for repositories related to the Kaggle APTOS 2019 blindness detection competition.

```
/* Task Definition Phase */

Search for GitHub repositories related to "aptos2019-blindness-detection".
Define evaluation criteria for repositories.
Specify JSON output format with repository ranking and recommendation.

/* Intent Analysis Phase */

Extract core entities: GitHub repositories, aptos2019-blindness-detection.
Identify keywords: "GitHub repositories", "solution approach", "performance".
Propose search queries: "aptos2019-blindness-detection GitHub repository".
Plan search and browsing strategy.

/* Searching Phase */
Perform query: "aptos2019-blindness-detection GitHub repository".
Retrieve initial repository list.

/* Browsing Phase */

browsing: {"query": "README", "url": "https://github.com/nitesh585/APTOS-2019-Blindness-Detection"}

browsing: {"query": "README", "url": "https://github.com/PrasunDatta/APTOS-2019-Blindness-Detection"}

browsing: {"query": "README", "url": "https://github.com/gdineshk6174/APTOS-2019-Blindness-Detection"}

browsing: {"query": "README", "url": "https://github.com/filipmu/Kaggle-APLOS-2019-Blindness-Detection"}

browsing: {"query": "README", "url": "https://github.com/tahsin314/40th_place_solution_aptos2019-blindness-detection"}

/* Analysing Phase */
Analyze search results to determine if there is sufficient information to answer the question:
1. nitesh585/APTOS-2019-Blindness-Detection
[Solution Approach]: Utilizes machine learning models for disease detection using retina images.
[Performance]: No specific performance metrics mentioned.
[Reason] : Focuses on the competition task with a machine learning approach, but lacks detailed performance data.
2. PrasunDatta/APTOS-2019-Blindness-Detection
[Solution Approach]: Aims to detect diabetic retinopathy using retina images, with a focus on automatic screening.
[Performance]: No specific performance metrics mentioned.
[Reason]: Provides a comprehensive approach but lacks detailed performance data.
3. gdineshk6174/APTOS-2019-Blindness-Detection
[Solution Approach]: Machine learning model to identify diabetic retinopathy automatically.
[Performance]: No specific performance metrics mentioned.
[Reason]: Focuses on automatic identification but lacks detailed performance data.
4. filipmu/Kaggle-APLOS-2019-Blindness-Detection
[Solution Approach]: Ensemble of deep learning classifiers (ResNet34, EfficientNetB6)
trained on original and preprocessed images.
```

```

[Performance]: Achieved a kappa metric of 0.904 on a test set.
[Reason]: Provides a detailed solution approach with specific performance metrics.
5. tahsin314/40th_place_solution_aptos2019-blindness-detection
[Solution Approach]: Information unavailable due to connection error.
[Performance]: Information unavailable.
[Reason]: Unable to retrieve information.

/* Results */
{
  "repositories": [
    {
      "rank": 1,
      "name": "filipmu/Kaggle-APTO-S-2019-Blindness",
      "url": "https://github.com/filipmu/Kaggle-APTO-S-2019-Blindness",
      "stars": "N/A",
      "can_solve_task": true,
      "solution_approach": "Ensemble of deep learning classifiers (ResNet34)",
      "performance": "Kappa metric of 0.904 on a test set.",
      "reason": "Provides a detailed solution approach with specific performance metrics."
    },
    {
      "rank": 2,
      "name": "nitesh585/APTO-S-2019-Blindness-Detection",
      "url": "https://github.com/nitesh585/APTO-S-2019-Blindness-Detection",
      "stars": "N/A",
      "can_solve_task": true,
      "solution_approach": "Machine learning models for disease detection using retinal images.",
      "performance": "Not specified.",
      "reason": "Focuses on the competition task with a machine learning approach, but lacks detailed performance metrics."
    },
    {
      "rank": 3,
      "name": "PrasunDatta/APTO-S-2019-Blindness-Detection",
      "url": "https://github.com/PrasunDatta/APTO-S-2019-Blindness-Detection",
      "stars": "N/A",
      "can_solve_task": true,
      "solution_approach": "Detect diabetic retinopathy using retina images with deep learning.",
      "performance": "Not specified.",
      "reason": "Provides a comprehensive approach but lacks detailed performance metrics."
    },
    {
      "rank": 4,
      "name": "gdineshk6174/APTO-S-2019-Blindness-Detection",
      "url": "https://github.com/gdineshk6174/APTO-S-2019-Blindness-Detection",
      "stars": "N/A",
      "can_solve_task": true,
      "solution_approach": "Machine learning model to identify diabetic retinopathy from retinal images.",
      "performance": "Not specified.",
      "reason": "Focuses on automatic identification but lacks detailed performance metrics."
    },
    {
      "rank": 5,
      "name": "tahsin314/40th_place_solution_aptos2019-blindness-detection",
      "url": "https://github.com/tahsin314/40th_place_solution_aptos2019-blindness-detection",
      "stars": "N/A",
      "can_solve_task": false,
      "solution_approach": "Information unavailable due to connection error."
    }
  ]
}

```

```

        "performance": "Information unavailable.",
        "reason": "Unable to retrieve information."
    }
],
"recommendation": "The repository 'filipmu/Kaggle-APTO-S-2019-Blindness' is
recommended as it provides a detailed solution approach with specific
performance metrics. Let me know if you need further assistance or if this
concludes your request."
}

```

The above log shows the results of our repository selection using the methodology in Section 3.1. For each competition task in our study, we followed a similar procedure to identify the most appropriate implementation repositories. The repositories were evaluated based on their documentation completeness, solution approach sophistication, and reported performance metrics when available.

C. Example of Task Prompt

Listing 3 presents the prompt for the Old Photo Restoration task, including the task requirements, the input/output formats and paths, and the repository used.

Listing 3: The prompt for old photo restoration.

```

# Task

I need you to analyze the following provided code repository and use your powerful
programming abilities to complete the user's task:

**Task Description**
<task>
Please complete the following tasks based on the given repository:

## Task Description
Given an old photo with scratches, remove its scratches.

## Available Repository
- **Name**: DeScratch
- **Path (absolute)**: /mnt/DeScratch_01/repo/DeScratch
- **Understanding Guide**: ['Read README.md to understand the project's basic
    functionality and usage']

## File Path
Input file paths and descriptions:
[
    {
        "path": "/mnt/DeScratch_01/repo/DeScratch/input_dataset/DeScratch_01_input.
            jpeg",
        "description": "An old photo in need of restoration"
    }
]

## Output
- **Output directory**: /mnt/DeScratch_01/repo/DeScratch/output_result
- If there is only one file, name it `output`.
- If there are multiple files, start naming with `output_01`, `output_02`, etc.,
    format as needed.

## Additional Notes
**Core Objective**: Quickly understand and analyze the code repository, generate
and execute the necessary code or tool calls to efficiently and accurately
complete the user's specified task.

**Core Components**:

```

```

- Saved output results must be named starting with `output` (e.g., `output.txt`, `output.wav`).
- If there are multiple files, name them `output_01`, `output_02`, etc.
- Ensure they are saved in a first-level subdirectory (e.g., `'{target_output_path}`/output.txt`), as subsequent tasks will match the results based on this field in that subdirectory for testing task completion effectiveness.
</task>

**Working Directory (where code should be run)**
<work_dir>
/mnt/DeScratch_01(repo/DeScratch
</work_dir>

**Repository Address**
<repo>
/mnt/DeScratch_01(repo/DeScratch
</repo>
```

D. Example of Initial Repository Content

Listing 4 presents the initial repository context for the DeScratch⁴ codebase used in the GitTaskBench dataset, serving as the agent's launchpad for understanding DeScratch's functionality and usage.

Listing 4: Initial repository context for the DeScratch.

```

**Overview of important parts of the code base**

----- (a) README.md -----
file_path: README.md
file_content:
# Old Photo Restoration (Official PyTorch Implementation)

The framework now supports the restoration of high-resolution input.

Training code is available and welcome to have a try and learn the training
details.

.....
## Requirement
The code is tested on Ubuntu with Nvidia GPUs and CUDA installed. Python>=3.6 is
required to run the code.

## Installation

Clone the Synchronized-BatchNorm-PyTorch repository for

.....
## :rocket: How to use?
**Note**: GPU can be set 0 or 0,1,2 or 0,2; use -1 for CPU

For images without scratches:
```
python run.py --input_folder [test_image_folder_path] \
 --output_folder [output_path] \
 --GPU 0
```
For scratched images:
```
python run.py --input_folder [test_image_folder_path] \

```

<sup>4</sup><https://github.com/microsoft/Bringing-Old-Photos-Back-to-Life>

```

--output_folder [output_path] \
--GPU 0 \
--with_scratch
```
.....
## License

The codes and the pretrained model in this repository are under the MIT license as
specified by the LICENSE file. We use our labeled dataset to train the
scratch detection model.
-----README.md-----
-----(b) Summary of key files-----
**Module Function Summary**



| Module Path                   | Function Description                                |
|-------------------------------|-----------------------------------------------------|
| run.py                        | Main entry point, orchestrates the restoration flow |
| GUI.py                        | Provides the graphical user interface entry point   |
| predict.py                    | Cog interface wrapper for the predictor             |
| Global/test.py                | Global image restoration entry point                |
| Face_Enhancement/test_face.py | Facial image enhancement testing                    |
| pix2pixHD_model.py            | Core model implementation                           |
| mapping_model.py              | Domain mapping model                                |
| networks.py                   | Neural network architecture definitions             |


-----Summary of key files-----
-----(c) Core components-----
```python
predict.py
class Predictor(cog.Predictor):
 def predict(self, image, HR=False, with_scratch=False):
 ...
```
```python
Global/models/pix2pixHD_model.py
class Pix2PixHDMModel(BaseModel):
 def inference(self, label, inst, image=None, feat=None):
 ...
```
-----Core components-----
-----(d) Other module paths-----
```text
Global/models/mapping_model.py
Global/data/online_dataset_for_old_photos.py
Face_Enhancement/models/pix2pix_model.py
Global/options/base_options.py
Face_Enhancement/util/util.py
Global/detection_util/util.py
Face_Enhancement/options/base_options.py
Global/detection_models/networks.py
Global/models/NonLocal_feature_mapping_model.py
Face_Enhancement/models/networks/architecture.py
Global/util/visualizer.py
Global/train_mapping.py
Global/train_domain_B.py
Global/train_domain_A.py
```
-----Other module paths-----

```

E. Detailed Experiment Execution Logs

This appendix presents the detailed execution logs for both the MLE-R and GitTaskBench experiments. These logs document the end-to-end experimental workflows, including task execution traces, error handling, repeated trials for failed steps, and verification of final outputs. The format follows top-tier conference standards, with line widths controlled for readability. All logs are presented in English.

E.1 Example Demonstration of MLE-R: Cancer Detection Experiment Log

The following log outlines the execution of a representative task from the MLE-R suite, specifically the histopathologic cancer detection task:

Listing 5: Cancer detection model training log.

```
User:  
I need you to analyze the following code repository and use your programming  
skills to complete the user task:  
  
Task Description:  
Kaggle Task: histopathologic-cancer-detection  
Overview:  
In this competition, you must create an algorithm to identify metastatic cancer in  
small image patches taken from larger digital pathology scans...  
[Full task and data description omitted for brevity]  
  
Working Directory:  
.....  
  
Repository Path:  
.....  
  
---  
  
Assistant:  
Thank you for the detailed information. I have understood the requirements and the  
structure of the repository. I will create a PyTorch-based script named `  
train_and_predict.py` for training and inference.  
  
---  
  
System:  
File not found: .....  
---  
  
Assistant:  
The file does not exist. Let's create it.  
  
---  
  
System:  
Error: Function edit_file not found.  
  
---  
  
Assistant:  
Sorry, I cannot directly edit files. Let's use Python's built-in functions to  
create and write the file.  
  
---  
  
User:  
[Python code to create `train_and_predict.py` is executed]
```

```
---
System:
Code output: File created and content written to .....
File successfully created.

---

Assistant:
Great! The file has been created. Let's run the script to start training and
prediction.

---

System:
exitcode: 1 (execution failed)
Code output: Traceback (most recent call last):
File ".../train_and_predict.py", line 195, in <module>
main()
File ".../train_and_predict.py", line 131, in main
train_dataset = CancerDataset(csv_file=train_data, root_dir=TRAIN_DIR, transform=
    transform)
...
TypeError: argument of type 'method' is not iterable
Using device: cuda

---

Assistant:
I see the issue: we passed a DataFrame instead of a CSV file path to the
CancerDataset class. Let's update the code accordingly.

---

[Assistant guides user through code updates using Python scripts to patch the file
.]

---

System:
Code output: File .../train_and_predict.py has been updated successfully.

---

Assistant:
Now let's try running the script again.

---

System:
exitcode: 1 (execution failed)
Code output: IndentationError: unexpected indent

---

Assistant:
We introduced an indentation error. Let's fix it with another script.

---

System:
Code output: File .../train_and_predict.py has been updated and indentation fixed.
Using device: cuda
Epoch 1/8, Train Loss: 0.3771, Val Loss: 0.2913, Val AUC: 0.9455
Epoch 2/8, Train Loss: 0.2883, Val Loss: 0.2473, Val AUC: 0.9614
```

```

Epoch 3/8, Train Loss: 0.2501, Val Loss: 0.2200, Val AUC: 0.9697
Epoch 4/8, Train Loss: 0.2246, Val Loss: 0.2143, Val AUC: 0.9718
Epoch 5/8, Train Loss: 0.2101, Val Loss: 0.2090, Val AUC: 0.9718
Epoch 6/8, Train Loss: 0.1978, Val Loss: 0.1912, Val AUC: 0.9771
Epoch 7/8, Train Loss: 0.1863, Val Loss: 0.2019, Val AUC: 0.9739
Epoch 8/8, Train Loss: 0.1774, Val Loss: 0.2126, Val AUC: 0.9766
Training and prediction completed. Results saved in 'result_submission.csv'
Result file exists and has the correct format.

---
Assistant:
Excellent! The script ran successfully, trained the model for 8 epochs, and
produced a valid result file. Validation AUC reached 0.9766. The workflow
included:
- Initial file creation and coding.
- Error handling and iterative code patching.
- Final successful execution and result verification.

```

E.2 Example Demonstration of GitTaskBench: Old Photo Scratch Removal Experiment Log

Below is the detailed log for an automated scratch removal task using the DeScratch codebase from the GitTaskBench suite. This log demonstrates the full process, including environment setup, error handling, dependency management, repeated attempts, and final verification:

Listing 6: Old photo scratch removal experiment log.

```

User:
Task: Remove scratches from an old photo using the DeScratch repository.
Input: task28/DeScratch_01/.../DeScratch/input_dataset/DeScratch_01_input.jpeg
Output directory: coding_run/task28/DeScratch_01/.../output_result

---
Assistant:
1. Check input image existence.
2. Set up environment variables.
3. Run main script to process image.
4. Verify output.

---
System:
Input file exists but is an image (cannot display directly).

---
Assistant:
Set PYTHONPATH and prepare environment.

---
System:
Environment setup completed.

---
Assistant:
Run main DeScratch processing script via subprocess.

---
System:

```

```
exitcode: 1 (execution failed)
Error: FileNotFoundError: No such file or directory: './Global'
Command output: Running Stage 1: Overall restoration

---
Assistant:
Likely not running in the correct directory. Change working directory to repo root
and rerun.

---
System:
exitcode: 0 (execution succeeded)
Command output:
You are using NL + Res
./checkpoints/restoration/VAE_A_quality/latest_net_G.pth not exists yet!
./checkpoints/restoration/VAE_B_scratch/latest_net_G.pth not exists yet!
./checkpoints/restoration/mapping_scratch/latest_net_mapping_net.pth not exists
yet!
Running Stage 1: Overall restoration
Finish Stage 1 ...
...
All the processing is done. Please check the results.
Image processing completed successfully.
Output files: []

---
Assistant:
Script ran, but output directory is empty. Warnings indicate missing pretrained
model checkpoints.

---
Assistant:
Check output folder structure, input files, and checkpoint existence.

---
System:
Output folder structure:
output_result/
    final_output/
    stage_2_detection_output/
    stage_3_face_output/
    stage_1_restore_output/
        origin/
        input_image/
        restored_image/
No image files found in output folder.
Input folder: ['.ipynb_checkpoints', 'DeScratch_01_input.jpeg']
Checkpoints missing.

---
Assistant:
README indicates pretrained models must be downloaded. Download and extract
required checkpoints.

---
System:
Checkpoints downloaded and extracted successfully. Zip files removed.
```

```

---
Assistant:
Rerun processing script.

---

System:
exitcode: 0 (execution succeeded)
Command output:
You are using NL + Res
Running Stage 1: Overall restoration
Finish Stage 1 ...
...
All the processing is done. Please check the results.
Image processing completed successfully.
Output files: []

---

Assistant:
Still no output files. Check output folder structure and run.log for clues.

---

System:
Output folder structure:
output_result/
    final_output/
    stage_2_detection_output/
    stage_3_face_output/
    stage_1_restore_output/
        origin/
        input_image/
        restored_image/
Input folder: ['.ipynb_checkpoints', 'DeScratch_01_input.jpeg']
run.log file not found

---

Assistant:
Try to directly call run.py processing steps for more debug output.

---

System:
exitcode: 0 (execution succeeded)
Traceback (most recent call last):
  File ".../DeScratch/Global/detection.py", line 17, in <module>
    from detection_models import networks
...
ModuleNotFoundError: No module named 'detection_models.sync_batchnorm'
...
ModuleNotFoundError: No module named 'dlib'
...
Output files: []
stage_1_restore_output: ['origin', 'input_image', 'restored_image']
stage_2_detection_output: []
stage_3_face_output: []

---

Assistant:
Missing Python modules: detection_models.sync_batchnorm, dlib, models.networks.
sync_batchnorm.

```

```
Install dependencies.

---

System:
All dependencies installed successfully.

---

Assistant:
Prepare input/output folders. Confirm input image exists.

---

System:
Files in input folder: ['.ipynb_checkpoints', 'DeScratch_01_input.jpeg']
Input and output folders prepared.

---

Assistant:
Rerun processing, focus on Stage 1.

---

System:
exitcode: 0 (execution succeeded)
Error executing detection.py: ModuleNotFoundError: No module named 'detection_models.sync_batchnorm'
Test output: You are using NL + Res
Now you are processing DeScratch_01_input.jpeg
Finish Stage 1 ...
output_result/
    final_output/
        DeScratch_01_input.jpeg
    stage_1_restore_output/
        origin/
            DeScratch_01_input.jpeg
        input_image/
            DeScratch_01_input.jpeg
        restored_image/
            DeScratch_01_input.jpeg
    masks/
        mask/
            DeScratch_01_input.png
    input/
        DeScratch_01_input.jpeg

---

Assistant:
Still missing sync_batchnorm. Clone Synchronized-BatchNorm-PyTorch and copy sync_batchnorm.

---

System:
Synchronized BatchNorm module installed successfully.

---

Assistant:
Rerun process, now error: ModuleNotFoundError: No module named 'easydict'.
Install easydict and other dependencies.
```

```
---  
System:  
Additional dependencies installed successfully.  
  
---  
  
Assistant:  
Rerun processing script with extra debug output.  
  
---  
  
System:  
exitcode: 0 (execution succeeded)  
Running Stage 1: Overall restoration  
Copied input image to .../masks/input  
Created blank mask at .../masks/mask/DeScratch_01_input.png  
Executing detection.py --test_path .../input_dataset ...  
Command output: initializing the dataloader  
model weights loaded  
processing DeScratch_01_input.jpeg  
Executing test.py --Scratch_and_Quality_restore ...  
Command output: You are using NL + Res  
Now you are processing DeScratch_01_input..png  
Now you are processing DeScratch_01_input.jpeg  
Copied DeScratch_01_input..png to final output directory  
Copied DeScratch_01_input.jpeg to final output directory  
Finish Stage 1 ...  
output_result/  
    final_output/  
        DeScratch_01_input..png  
        DeScratch_01_input.jpeg  
    stage_1_restore_output/  
        origin/  
            DeScratch_01_input..png  
            DeScratch_01_input.jpeg  
        input_image/  
            DeScratch_01_input..png  
            DeScratch_01_input.jpeg  
        restored_image/  
            DeScratch_01_input..png  
            DeScratch_01_input.jpeg  
    masks/  
        mask/  
            DeScratch_01_input.png  
            DeScratch_01_input..png  
    input/  
        DeScratch_01_input..png  
        DeScratch_01_input.jpeg  
  
---  
  
Assistant:  
Processing succeeded. Compare input and output images.  
  
---  
  
System:  
Comparison image saved as 'image_comparison.png'  
Input image size: (559, 878)  
Output image size: (556, 876)  
Input image format: JPEG  
Output image format: JPEG
```

Assistant:
 Processing completed. Input and output images are available for visual comparison.
 The pipeline handled errors through iterative debugging, resolved all missing dependencies, and produced restored images as required.

These logs collectively demonstrate the robustness and reproducibility of our experimental framework. All steps, including failures and subsequent recovery, are documented to ensure transparency and facilitate future benchmarking.

F. Details of Module-Level Scoring Features

Dependency. As shown in Algorithm 1, to measure a module’s centrality in the module dependency graph, we apply Personalized PageRank, emphasizing the target module with a personalized vector. The algorithm iteratively computes the PageRank score using a damping factor of $\alpha = 0.85$, and the resulting score is scaled for comparability with other metrics.

Complexity. The feature calculates the code complexity of a module by evaluating two core factors: the frequency of branching and looping structures, and the depth of function nesting. The occurrence frequency of branching and looping structures (such as `if`, `for`, `while`, and `except`) is normalized to prevent the complexity score from becoming excessively high. The function nesting depth is quantified based on the indentation level, and the resulting score is scaled by a proportional factor. The final complexity score is a composite of these two factors, with each factor’s score normalized to a maximum value of 1.0, providing a comprehensive and balanced measure of the module’s code complexity, as described in Algorithm 2.

Usage. This feature calculates the number of times other modules import the current module, as well as the number of times the functions and classes within the module are called, using these metrics as indicators of the module’s significance.

Semantic. we conduct a semantic importance analysis based on the naming conventions of modules to identify key files. Commonly recognized names such as `__init__`, `app`, `settings`, `config`, `utils`, and `constants` are considered to be of higher significance. In addition to this, we examine the presence of crucial semantic keywords within the code, including terms like `main`, `core`, `engine`, `api`, `service`, `processor`, `factory`, `builder`, `scheduler`, and `config`. The occurrence of these keywords in module or function names serves as an important factor in determining the module’s relevance and is used as a scoring criterion in our analysis.

Doc. This feature calculates a score based on the existence and quality of the module’s docstring to assess the completeness of the module’s documentation. The score is initially determined by the length of the docstring. Additionally, the feature checks for the presence of specific sections within the docstring, such as descriptions of parameters, return values, and examples. If the first two sections are present, the score increases for each, and if examples are included, the score also increases. This approach provides a comprehensive evaluation of the documentation’s completeness.

Git. This feature calculates whether the file is part of a Git repository or not. If the file is not in a Git repository, a score of 0.0 is returned. Next, it counts the number of commits associated with the file. The score is initially based on the number of commits. The feature also takes into account the time of the last commit, and recently modified files are considered more important, with the time factor accounting for 30% of the final score. The final importance score is a weighted combination of the number of submissions and the time-of-submission score to ensure that both the frequency of modifications and the freshness of the file are taken into account.

Table 4: Token Usage by Agent Across LLMs on MLE-R (in Millions of Tokens).

| Agent | GPT-4o (M) | Claude 3.5 (M) | Deepseek v3 (M) |
|------------|------------|----------------|-----------------|
| Openhands | 5.26 | 4.73 | 4.93 |
| SWE-Agent | 1.29 | 1.51 | 1.48 |
| RepoMaster | 0.36 | 0.32 | 0.18 |

Algorithm 1 Calculate Personalized PageRank Score for a Module

```
1: Input: Module dependency graph  $G_m$ , target module  $m$ 
2: Output: Personalized PageRank score  $score$ 
3:  $score \leftarrow 0.0$ 
4: if  $|V(G_m)| > 0$  then
5:    $personalization[n] \leftarrow 1.0$  for all  $n \in V(G_m)$ 
6:    $personalization[m] \leftarrow 2.0$ 
7:    $pagerank \leftarrow \text{PageRank}(G_m, \alpha = 0.85, personalization)$ 
8:    $score \leftarrow pagerank[moduleId] \times 10$ 
9: end if
10: return  $score$ 
```

Algorithm 2 Analyze Code Complexity of a Module

```
1: Input: target module  $m$ 
2: Output: Code complexity score  $score$ 
3:  $score \leftarrow 0.0$ 
4:  $content \leftarrow m.content$ 
5:  $branchCount \leftarrow \sum_{k \in \{if,for,while,except\}} \text{count}(k, \text{in } content)$ 
6:  $score \leftarrow \min(branchCount/50.0, 1.0)$ 
7:  $matches \leftarrow \text{count}(def, \text{in } content)$ 
8: if  $matches \neq \emptyset$  then
9:    $indentLevel \leftarrow \max(matches)/4$ 
10:   $score \leftarrow score + \min(indentLevel/5.0, 1.0) \times 0.3$ 
11: end if
12: return  $score$ 
```

G. Detailed Experimental Results and Analysis

G.1 Detailed Results on MLE-R

To ensure transparency and facilitate reproducibility, we present the complete raw results of our experiments in this section. Tables 6, 7 and 8 present RepoMaster’s scores on all 22 MLE-R tasks when paired with GPT-4o, Claude 3.5, and Deepseek v3, respectively, reporting for each task the achieved score, relevant medal-category thresholds, and binary indicators of whether performance criteria are met, thereby enabling detailed comparisons between models and supporting further analysis of their capabilities. Additionally, Tables 9 and 10 show the results of Openhands and SWE-Agent on MLE-R with Claude 3.5, respectively.

Table 4 presents the token usage of each agent across GPT-4o, Claude 3.5, and Deepseek v3 when handling tasks from the MLE-R dataset. It shows that Openhands consumes the most tokens (around 5M per task), SWE-Agent uses a moderate amount (1.3–1.5M), and RepoMaster is by far the most efficient (0.18–0.36M), with all three agents exhibiting a slight reduction in token usage when moving from GPT-4o to Claude-3.5 to Deepseek V3; this pattern highlights how model-specific tokenization and interaction protocols affect consumption, with RepoMaster’s lightweight design offering significant cost savings for large-scale MLE-R tasks.

G.2 Detailed Results on GitTaskBench

We report the detailed results (especially the number of tokens) of different frameworks with different models on GitTaskBench in Tables 5.

G.3 Ablation Study

To more intuitively represent the role of each core component, we present the results of the Ablation Study visually, as shown in Figure 4.

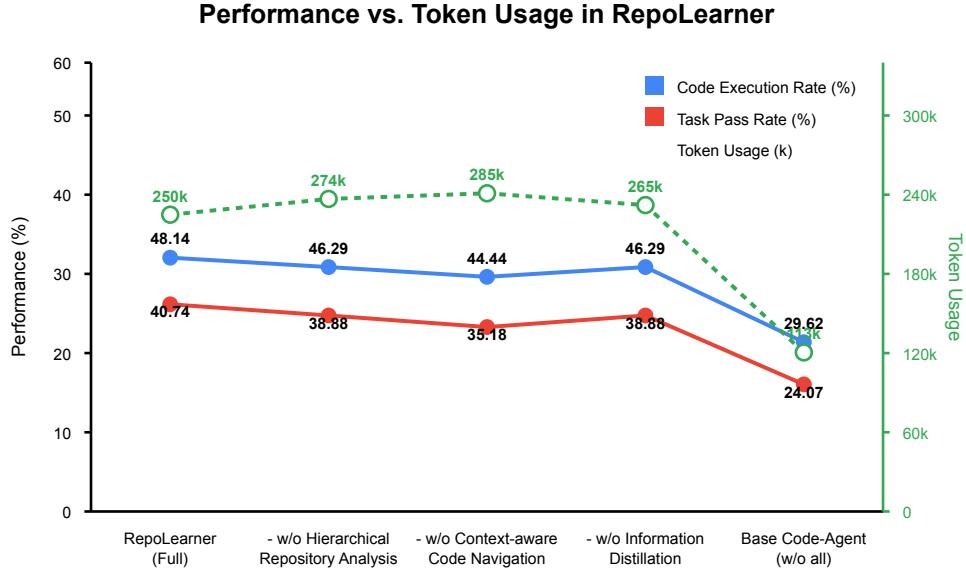


Figure 4: Performance comparison of different ablation configurations in RepoMaster.

Table 5: Comparison of Different Frameworks on GitTaskBench.

| Framework | LLM | Execution Completion Rate (%) | Task Pass Rate (%) | #Total Tokens |
|------------|-------------|-------------------------------|--------------------|---------------|
| SWE-Agent | GPT-4o | 29.63 | 9.26 | 308155 |
| | Claude 3.5 | 44.44 | 14.81 | 330676 |
| | DeepSeek V3 | 29.63 | 9.26 | 265538 |
| OpenHands | GPT-4o | 37.04 | 14.81 | 1195312 |
| | Claude 3.5 | 48.15 | 24.07 | 3094878 |
| | DeepSeek V3 | 42.59 | 16.67 | 7662130 |
| RepoMaster | GPT-4o | 48.14 | 40.74 | 250245 |
| | Claude 3.5 | 75.92 | 62.96 | 154913 |
| | DeepSeek V3 | 61.11 | 44.44 | 255998 |

Table 6: RepoMaster with GPT-4o: Raw Results.

| Competition | Score | Gold Threshold | Silver Threshold | Bronze Threshold | Median Threshold | Any Medal | Gold Medal | Silver Medal | Bronze Medal | Above Median | Valid Sub. |
|---|---------|----------------|------------------|------------------|------------------|-----------|------------|--------------|--------------|--------------|------------|
| aerial-cactus-identification | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.99910 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| aptos2019-blindness-detection | 0.89501 | 0.93051 | 0.91965 | 0.91449 | 0.88891 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| chaii-hindi-and-tamil-question-answering | 0.68616 | 0.76305 | 0.73984 | 0.73725 | 0.72756 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| denoising-dirty-documents | - | 0.01794 | 0.02609 | 0.04517 | 0.07325 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dog-breed-identification | 0.72540 | 0.00050 | 0.00539 | 0.04598 | 0.47205 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| dogs-vs-cats-redux-kernels-edition | 0.10353 | 0.03882 | 0.05038 | 0.06127 | 0.12216 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| jigsaw-toxic-comment-classification-challenge | 0.98639 | 0.98740 | 0.98668 | 0.98639 | 0.98079 | TRUE | FALSE | FALSE | TRUE | TRUE | TRUE |
| histopathologic-cancer-detection | 0.99112 | 0.98350 | 0.97980 | 0.97380 | 0.94770 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| leaf-classification | 0.04422 | 0.00000 | 0.00791 | 0.01526 | 0.10835 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| learning-agency-lab-automated-essay-scoring-2 | 0.78242 | 0.83583 | 0.83518 | 0.83471 | 0.82827 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| lmsys-chatbot-arena | 1.08268 | 0.98392 | 0.99277 | 1.00283 | 1.05212 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| mlsp-2013-birds | - | 0.93527 | 0.90038 | 0.87372 | 0.86572 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| new-york-city-taxi-fare-prediction | 4.76987 | 2.83377 | 2.88191 | 2.92371 | 3.59741 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| nomad2018-predict-transparent-conductors | 0.06692 | 0.05589 | 0.06229 | 0.06582 | 0.06988 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| plant-pathology-2020-fgvc7 | 0.94049 | 0.97836 | 0.97465 | 0.97361 | 0.94852 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| random-acts-of-pizza | 0.63818 | 0.97908 | 0.76482 | 0.69210 | 0.59960 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| spooky-author-identification | - | 0.16506 | 0.26996 | 0.29381 | 0.41879 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| tabular-playground-series-dec-2021 | 0.95711 | 0.95660 | 0.95658 | 0.95658 | 0.95342 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| ranzcr-clip-catheter-line-classification | 0.84915 | 0.97357 | 0.97152 | 0.97090 | 0.96750 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-may-2022 | 0.95056 | 0.99823 | 0.99822 | 0.99818 | 0.97268 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| text-normalization-challenge-russian-language | 0.00000 | 0.99012 | 0.98232 | 0.97592 | 0.97591 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| tgs-salt-identification-challenge | 0.52210 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |

Table 7: RepoMaster with Claude 3.5: Raw Results.

| Competition | Score | Gold Threshold | Silver Threshold | Bronze Threshold | Median Threshold | Any Medal | Gold Medal | Silver Medal | Bronze Medal | Above Median | Valid Sub. |
|---|---------|----------------|------------------|------------------|------------------|-----------|------------|--------------|--------------|--------------|------------|
| aerial-cactus-identification | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.99910 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| aptos2019-blindness-detection | 0.90685 | 0.93051 | 0.91965 | 0.91449 | 0.88891 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| chaii-hindi-and-tamil-question-answering | 0.67455 | 0.76305 | 0.73984 | 0.73725 | 0.72756 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| denoising-dirty-documents | - | 0.01794 | 0.02609 | 0.04517 | 0.07325 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dog-breed-identification | 0.58826 | 0.00050 | 0.00539 | 0.04598 | 0.47205 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| dogs-vs-cats-redux-kernels-edition | 0.06683 | 0.03882 | 0.05038 | 0.06127 | 0.12216 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| jigsaw-toxic-comment-classification-challenge | 0.98929 | 0.98740 | 0.98668 | 0.98639 | 0.98079 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| histopathologic-cancer-detection | 0.99254 | 0.98350 | 0.97980 | 0.97380 | 0.94770 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| leaf-classification | 2.02279 | 0.00000 | 0.00791 | 0.01526 | 0.10835 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| learning-agency-lab-automated-essay-scoring-2 | 0.78964 | 0.83583 | 0.83518 | 0.83471 | 0.82827 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| lmsys-chatbot-arena | 2.26734 | 0.98392 | 0.99277 | 1.00283 | 1.05212 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| mlsp-2013-birds | 0.65981 | 0.93527 | 0.90038 | 0.87372 | 0.86572 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| new-york-city-taxi-fare-prediction | 5.12441 | 2.83377 | 2.88191 | 2.92371 | 3.59741 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| nomad2018-predict-transparent-conductors | 0.06479 | 0.05589 | 0.06229 | 0.06582 | 0.06988 | TRUE | FALSE | FALSE | TRUE | TRUE | TRUE |
| plant-pathology-2020-fgvc7 | 0.98888 | 0.97836 | 0.97465 | 0.97361 | 0.94852 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| random-acts-of-pizza | 0.63403 | 0.97908 | 0.76482 | 0.69210 | 0.59960 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| spooky-author-identification | 0.50796 | 0.16506 | 0.26996 | 0.29381 | 0.41879 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-dec-2021 | 0.96227 | 0.95660 | 0.95658 | 0.95658 | 0.95342 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| ranzcr-clip-catheter-line-classification | 0.91278 | 0.97357 | 0.97152 | 0.97090 | 0.96750 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-may-2022 | 0.99373 | 0.99823 | 0.99822 | 0.99818 | 0.97268 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| text-normalization-challenge-russian-language | 0.18164 | 0.99012 | 0.98232 | 0.97592 | 0.97591 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| tgs-salt-identification-challenge | 0.52210 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |

Table 8: RepoMaster with Deepseek V3: Raw Results.

| Competition | Score | Gold Threshold | Silver Threshold | Bronze Threshold | Median Threshold | Any Medal | Gold Medal | Silver Medal | Bronze Medal | Above Median | Valid Sub. |
|---|------------|----------------|------------------|------------------|------------------|-----------|------------|--------------|--------------|--------------|------------|
| aerial-cactus-identification | 0.99988 | 1.00000 | 1.00000 | 1.00000 | 0.99910 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| aptos2019-blindness-detection | 0.87334 | 0.93051 | 0.91965 | 0.91449 | 0.88891 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| chaii-hindi-and-tamil-question-answering | 0.31169 | 0.76305 | 0.73984 | 0.73725 | 0.72756 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| denoising-dirty-documents | - | 0.01794 | 0.02609 | 0.04517 | 0.07325 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dog-breed-identification | 0.84673 | 0.00050 | 0.00539 | 0.04598 | 0.47205 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| dogs-vs-cats-redux-kernels-edition | 0.04717 | 0.03882 | 0.05038 | 0.06127 | 0.12216 | TRUE | FALSE | TRUE | FALSE | TRUE | TRUE |
| histopathologic-cancer-detection | 0.99300 | 0.98350 | 0.97980 | 0.97380 | 0.94770 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| jigsaw-toxic-comment-classification-challenge | 0.96374 | 0.98740 | 0.98668 | 0.98639 | 0.98079 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| leaf-classification | 0.07526 | 0.00000 | 0.00791 | 0.01526 | 0.10835 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| learning-agency-lab-automated-essay-scoring-2 | 0.89558 | 0.83583 | 0.83518 | 0.83471 | 0.82827 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| lmsys-chatbot-arena | 1.09199 | 0.98392 | 0.99277 | 1.00283 | 1.05212 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| mlsp-2013-birds | - | 0.93527 | 0.90038 | 0.87372 | 0.86572 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| new-york-city-taxi-fare-prediction | 1064.68701 | 2.83377 | 2.88191 | 2.92371 | 3.59741 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| nomad2018-predict-transparent-conductors | 0.06468 | 0.05589 | 0.06229 | 0.06582 | 0.06988 | TRUE | FALSE | FALSE | TRUE | TRUE | TRUE |
| plant-pathology-2020-fgvc7 | 0.92653 | 0.97836 | 0.97465 | 0.97361 | 0.94852 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| random-acts-of-pizza | 0.59387 | 0.97908 | 0.76482 | 0.69210 | 0.59960 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| ranzcr-clip-catheter-line-classification | - | 0.97357 | 0.97152 | 0.97090 | 0.96750 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| spooky-author-identification | 0.62857 | 0.16506 | 0.26996 | 0.29381 | 0.41879 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-dec-2021 | 0.96084 | 0.95660 | 0.95658 | 0.95658 | 0.95342 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-may-2022 | 0.99161 | 0.99823 | 0.99822 | 0.99818 | 0.97268 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| text-normalization-challenge-russian-language | 0.01451 | 0.99012 | 0.98232 | 0.97592 | 0.97591 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| tgs-salt-identification-challenge | 0.50440 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |

Table 9: Openhands with Claude 3.5: Raw Results.

| Competition | Score | Gold Threshold | Silver Threshold | Bronze Threshold | Median Threshold | Any Medal | Gold Medal | Silver Medal | Bronze Medal | Above Median | Valid Sub. |
|---|----------|----------------|------------------|------------------|------------------|-----------|------------|--------------|--------------|--------------|------------|
| aerial-cactus-identification | 0.99951 | 1.00000 | 1.00000 | 1.00000 | 0.99910 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| aptos2019-blindness-detection | - | 0.93051 | 0.91965 | 0.91449 | 0.88891 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| chhai-hindi-and-tamil-question-answering | - | 0.76305 | 0.73984 | 0.73725 | 0.72756 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| denoising-dirty-documents | - | 0.01794 | 0.02609 | 0.04517 | 0.07325 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dog-breed-identification | - | 0.00050 | 0.00539 | 0.04598 | 0.47205 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dogs-vs-cats-redux-kernels-edition | - | 0.03882 | 0.05038 | 0.06127 | 0.12216 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| histopathologic-cancer-detection | - | 0.98350 | 0.97980 | 0.97380 | 0.94770 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| jigsaw-toxic-comment-classification-challenge | - | 0.98740 | 0.98668 | 0.98639 | 0.98079 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| leaf-classification | 4.59036 | 0.00000 | 0.00791 | 0.01526 | 0.10835 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| learning-agency-lab-automated-essay-scoring-2 | 0.00000 | 0.83583 | 0.83518 | 0.83471 | 0.82827 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| lmsys-chatbot-arena | - | 0.98392 | 0.99277 | 1.00283 | 1.05212 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| mlsp-2013-birds | - | 0.93527 | 0.90038 | 0.87372 | 0.86572 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| new-york-city-taxi-fare-prediction | 13.42166 | 2.83377 | 2.88191 | 2.92371 | 3.59741 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| nomad2018-predict-transparent-conductors | 0.23121 | 0.05589 | 0.06229 | 0.06582 | 0.06988 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| plant-pathology-2020-fgvc7 | - | 0.97836 | 0.97465 | 0.97361 | 0.94852 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| random-acts-of-pizza | 0.54052 | 0.97908 | 0.76482 | 0.69210 | 0.59960 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| ranzcr-clip-catheter-line-classification | - | 0.97357 | 0.97152 | 0.97090 | 0.96750 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| spooky-author-identification | 0.92421 | 0.16506 | 0.26996 | 0.29381 | 0.41879 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| tabular-playground-series-dec-2021 | 0.95837 | 0.95660 | 0.95658 | 0.95658 | 0.95342 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-may-2022 | 0.93330 | 0.99823 | 0.99822 | 0.99818 | 0.97268 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| text-normalization-challenge-english-language | - | 0.99012 | 0.98232 | 0.97592 | 0.97591 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| tgs-salt-identification-challenge | 0.44300 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |

Table 10: SWE-Agent with Claude 3.5: Raw Results.

| Competition | Score | Gold Threshold | Silver Threshold | Bronze Threshold | Median Threshold | Any Medal | Gold Medal | Silver Medal | Bronze Medal | Above Median | Valid Sub. |
|---|---------|----------------|------------------|------------------|------------------|-----------|------------|--------------|--------------|--------------|------------|
| aerial-cactus-identification | 0.99917 | 1.00000 | 1.00000 | 1.00000 | 0.99910 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| aptos2019-blindness-detection | - | 0.93051 | 0.91965 | 0.91449 | 0.88891 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| chhai-hindi-and-tamil-question-answering | 0.04985 | 0.76305 | 0.73984 | 0.73725 | 0.72756 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| denoising-dirty-documents | - | 0.01794 | 0.02609 | 0.04517 | 0.07325 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dog-breed-identification | - | 0.00050 | 0.00539 | 0.04598 | 0.47205 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| dogs-vs-cats-redux-kernels-edition | - | 0.03882 | 0.05038 | 0.06127 | 0.12216 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| histopathologic-cancer-detection | - | 0.98350 | 0.97980 | 0.97380 | 0.94770 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| jigsaw-toxic-comment-classification-challenge | 0.97251 | 0.98740 | 0.98668 | 0.98639 | 0.98079 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| leaf-classification | 2.74648 | 0.00000 | 0.00791 | 0.01526 | 0.10835 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| learning-agency-lab-automated-essay-scoring-2 | - | 0.83583 | 0.83518 | 0.83471 | 0.82827 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| lmsys-chatbot-arena | - | 0.98392 | 0.99277 | 1.00283 | 1.05212 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| mlsp-2013-birds | 0.69828 | 0.93527 | 0.90038 | 0.87372 | 0.86572 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| new-york-city-taxi-fare-prediction | - | 2.83377 | 2.88191 | 2.92371 | 3.59741 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| nomad2018-predict-transparent-conductors | 0.09346 | 0.05589 | 0.06229 | 0.06582 | 0.06988 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| plant-pathology-2020-fgvc7 | 0.74045 | 0.97836 | 0.97465 | 0.97361 | 0.94852 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| random-acts-of-pizza | 0.64037 | 0.97908 | 0.76482 | 0.69210 | 0.59960 | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE |
| ranzcr-clip-catheter-line-classification | - | 0.97357 | 0.97152 | 0.97090 | 0.96750 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| spooky-author-identification | 1.08483 | 0.16506 | 0.26996 | 0.29381 | 0.41879 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| tabular-playground-series-dec-2021 | 0.95746 | 0.95660 | 0.95658 | 0.95658 | 0.95342 | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| tabular-playground-series-may-2022 | - | 0.99823 | 0.99822 | 0.99818 | 0.97268 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| text-normalization-challenge-english-language | - | 0.99012 | 0.98232 | 0.97592 | 0.97591 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| tgs-salt-identification-challenge | 0.06620 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |