



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

Huacan Wang ^{*†}
Midea Group
wanghc141@midea.com

Ziyi Ni ^{*}
The Key Laboratory of Cognition and Decision Intelligence for Complex Systems
School of Artificial Intelligence, University of Chinese Academy of Science
Institute of Automation, Chinese Academy of Sciences
niziyi2021@ia.ac.cn

Shuo Zhang ^{*}
School of Artificial Intelligence,
Beijing University of Posts and Telecommunications
shuoz@bupt.edu.cn

Shuo Lu
Institute of Automation,
Chinese Academy of Sciences
lushuo2024@ia.ac.cn

Sen Hu
Peking University
husen@pku.edu.cn

Ziyang He
National University of Singapore
e1553228@u.nus.edu

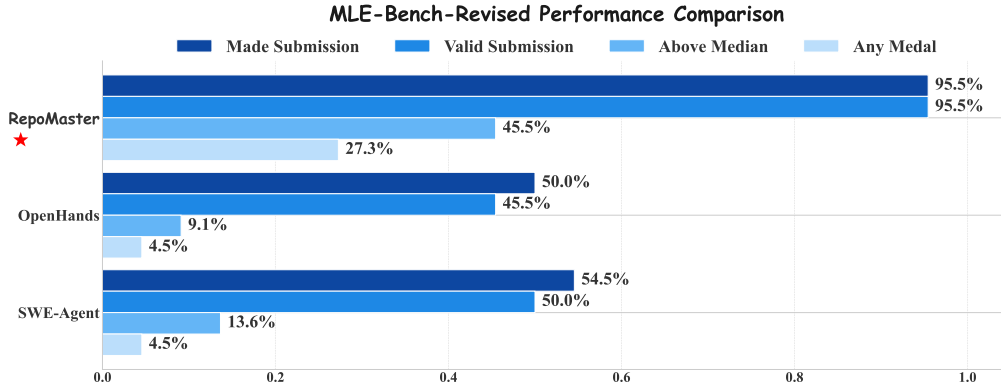
Chen Hu
StepFun
hatcher@stepfun.com

Jiaye Lin
Tsinghua University
lin-jy22@mails.tsinghua.edu.cn

Yifu Guo
University of Aberdeen
u08yg22@abdn.ac.uk

Yuntao Du [†]
C-FAIR, Shandong University
State Key Lab. for Novel Software Technology, NJU
yuntaodu@sdu.edu.cn

Pin Lyu [†]
Institute of Automation,
Chinese Academy of Sciences
pin.lv@ia.ac.cn



^{*}Equal contribution.

[†]Corresponding authors.

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], 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.

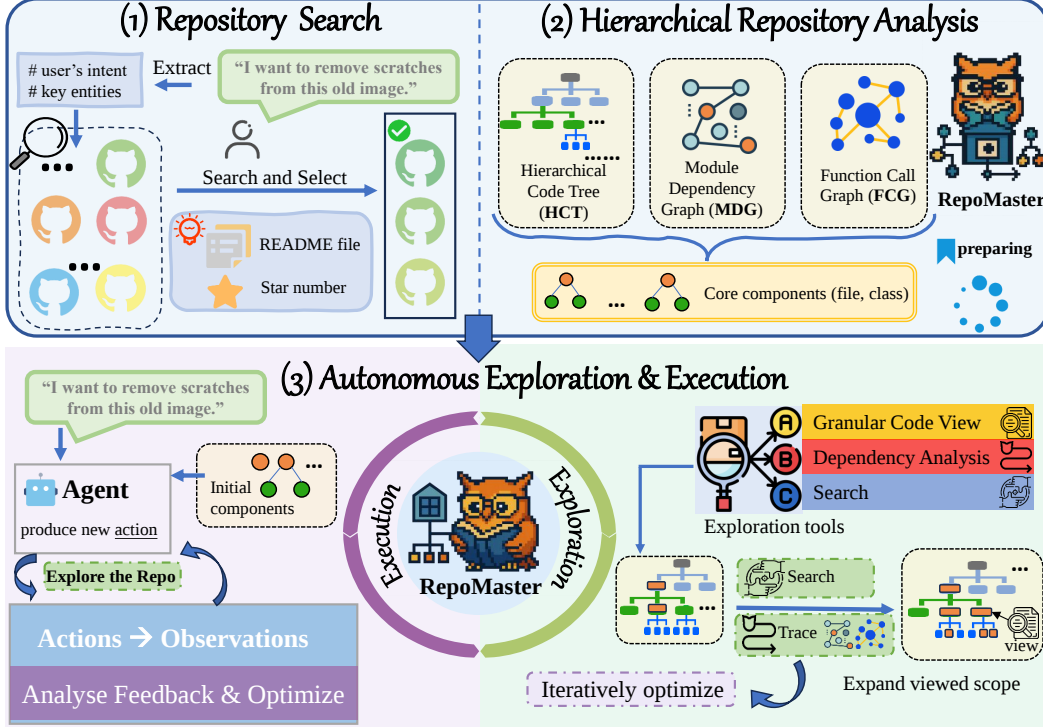


Figure 1: Overview of RepoMaster, consisting of Repository Search, Hierarchical Repository Analysis and Autonomous Exploration & Execution.

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. 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 $\rightarrow$ module $\rightarrow$ class $\rightarrow$ 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. The equal weighting scheme ($w_i \equiv 1$) was empirically validated through sensitivity analysis (see Appendix G.4 for details).

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.

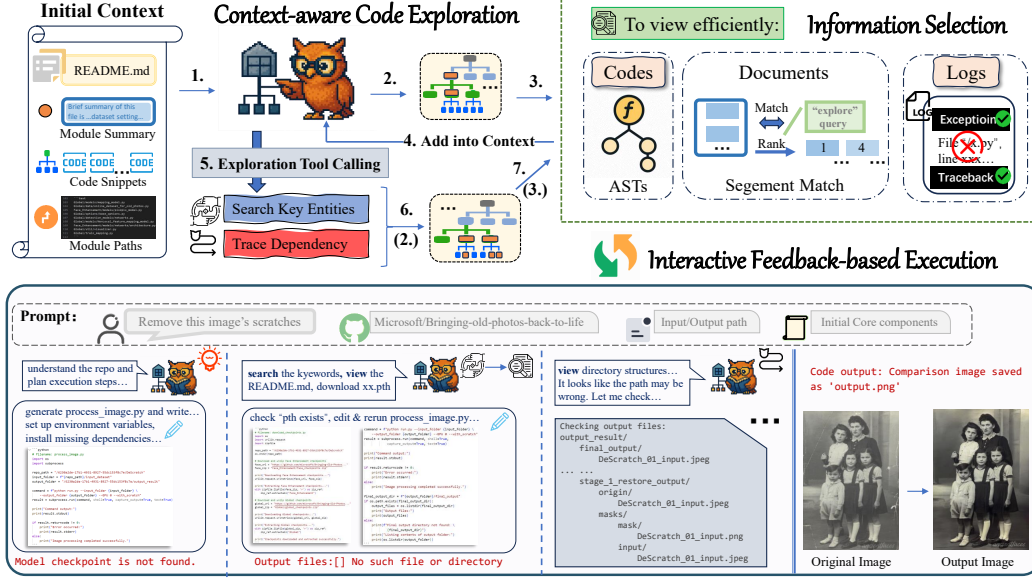


Figure 2: Overview of RepoMaster’s autonomous exploration–execution loop and an example demonstration. The agent begins by analyzing the initial context (Step 1) and specifies a file to inspect (Step 2). For efficient viewing, it extracts only the key information from that file (Step 3) and appends it 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.

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.

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 comparison³; 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 thresholds⁴ 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]⁵ 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.

³Ensure a fair comparison, as other general agent frameworks do not support automatic repository retrieval.

⁴The specific thresholds for gold, silver, and bronze medals are provided in Appendix G.

⁵More 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	<u>13.64</u>	18.18
	Claude 3.5	95.45	95.45	45.45	4.55	0.00	22.73	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>

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.

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	250k
	Claude 3.5	75.92	62.96	154k
	DeepSeek V3	61.11	44.44	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.

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

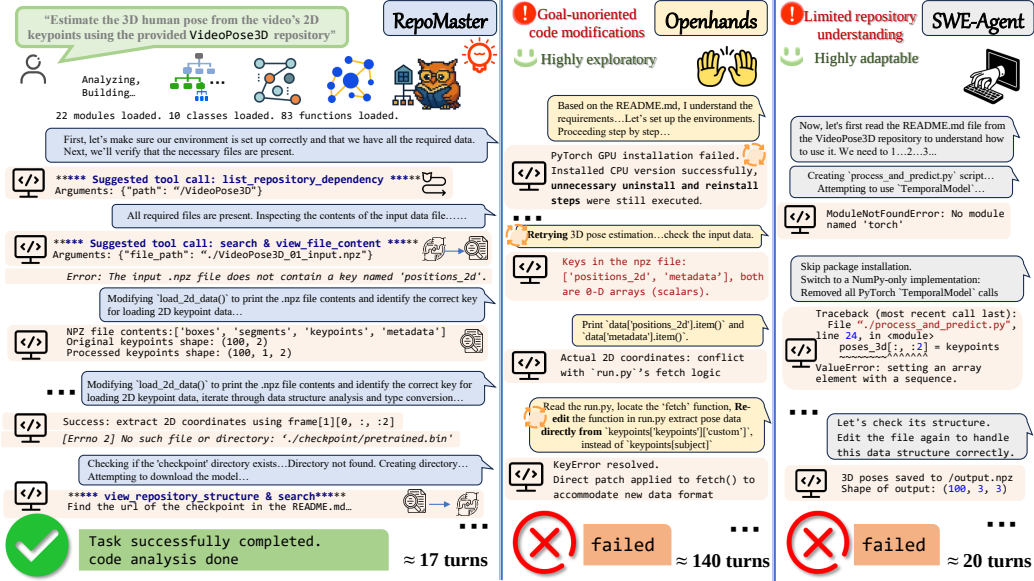


Figure 3: RepoMaster vs. OpenHands and SWE-Agent: A Comparative Case Study.

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.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant 62301559 and Shandong Provincial Natural Science Foundation (Grant No. ZR2025QC1570).

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. Ml-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, Dixing Xu, Ian Kaplan, Deniss Jacenko, 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 Mireshtgallah, Andrew Wang, Jiayi Gao, Wanjuan 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 Altschmidt, 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 Chakraborty, 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.
- [56] Ziyi Ni, Huacan Wang, Shuo Zhang, Shuo Lu, Ziyang He, Wang You, Zhenheng Tang, Yuntao Du, Bill Sun, Hongzhang Liu, Sen Hu, Ronghao Chen, Bo Li, Xin Li, Chen Hu, Binxing Jiao, Daxin Jiang, and Pin Lyu. Gittaskbench: A benchmark for code agents solving real-world tasks through code repository leveraging, 2025.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [NA].
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes]" is generally preferable to "[No]", it is perfectly acceptable to answer "[No]" provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No]" or "[NA]" is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading “NeurIPS Paper Checklist”,**
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers.**

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [Yes]

Justification: The paper claims main contributions and scope in the abstract and introduction.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Yes, the paper discusses the limitations of the work.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Our paper mainly works on method and experiments and do not include complete proof about the method.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: the paper will open source code on github for other researchers to reproduce experiments.

Guidelines:

- The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [\[Yes\]](#)

Justification: the paper will provide the open source code.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [\[Yes\]](#)

Justification: Yes, the paper used open sourced dataset and include all the details of the datasets.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [\[Yes\]](#)

Justification: Yes, the paper reported error bars properly.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [\[Yes\]](#)

Justification: Yes, the paper includes all the computational resources information.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: Yes, the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Yes, the paper discussed the issue.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: We mainly use open sourced models and commercialized model API. These models commonly have safeguards in their own license description.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [\[Yes\]](#)

Justification: Yes, our open sourced code will be licensed properly.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: We introduced a new method for the problem and will release our code on github if accepted.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Our paper does experiment on existing open sourced dataset and the evaluation code of the method is on github.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: We did not crowdsourcing our research experiments.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Our framework conducts a detailed comparison of the use of different LLMs, and the performance of the method does not rely on the LLMs themselves.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.

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.

More details are provided in [56].

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-APTOS-2019-Blindness"}

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-APTOS-2019-Blindness
 [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-APTOS-2019-Blindness",
      "url": "https://github.com/filipmu/Kaggle-APTOS-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/APTOS-2019-Blindness-Detection",
      "url": "https://github.com/nitesh585/APTOS-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/APTOS-2019-Blindness-Detection",
      "url": "https://github.com/PrasunDatta/APTOS-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/APTOS-2019-Blindness-Detection",
      "url": "https://github.com/gdineshk6174/APTOS-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-APTOS-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 DeScratch⁶ 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] \

```

<sup>6</sup><https://github.com/microsoft/Bringing-Old-Photos-Back-to-Life>

```

 --GPU 0
 ...
For scratched images:
 ...
python run.py --input_folder [test_image_folder_path] \
 --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 Pix2PixHDModel(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.

F.1 Computational Cost of Hierarchical Repository Analysis

Our static analysis preprocessing pipeline utilizes highly optimized libraries (e.g., tree-sitter for AST parsing) and supports multi-threaded execution.

For typical repositories (1-3k files): The entire preprocessing completes on average within 4.8 seconds, including:

- AST construction: ~ 2.1 seconds (parallelized across CPU cores)
- Dependency/call graph generation: ~ 1.9 seconds
- Core component identification: ~ 0.8 seconds

For complex repositories: For repositories where core components exceed the context threshold (8k tokens), we employ an LLM-based summarization approach using DeepSeek V3, which takes approximately 10 seconds. Even for complex repositories, overall preprocessing time remains within 15 seconds.

F.2 Weight Configuration for Importance Scoring

The weights in our importance scoring scheme (Equation 2) are set to $w_i = 1$ (equal weighting) for simplicity and interpretability. This configuration was determined through empirical analysis rather than exhaustive optimization. Our experiments show that while different weight combinations may affect the initial recall rate of important files, the impact on overall end-to-end task success is minimal (see Appendix G.4 for detailed sensitivity analysis). The equal weighting scheme offers a good balance between performance and simplicity, avoiding the need for task-specific tuning.

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 8, 9 and 10 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 11 and 12 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.

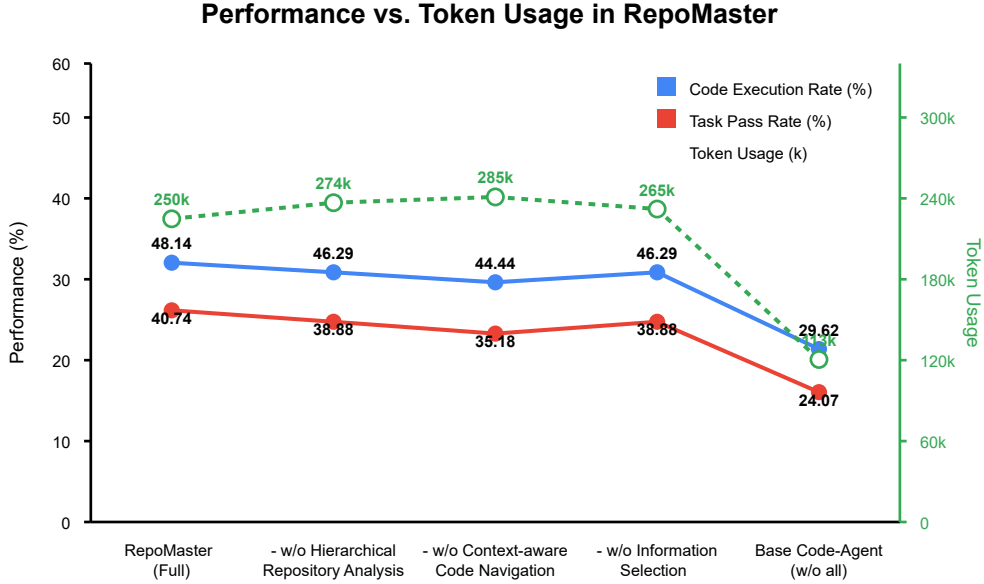


Figure 4: Performance comparison of different ablation configurations in RepoMaster.

G.4 Hyperparameter Selection and Sensitivity Analysis

This section provides detailed experimental validation for the hyperparameter choices used in RepoMaster’s core component identification (Section 3.2.2) and context initialization strategies.

G.4.1 Top-k Module Selection To determine the optimal number of top-ranked modules to include in the initial context, we manually constructed core file modules from several repositories as our test

Table 5: Comparison of Different Frameworks on GitTaskBench.

| Framework | LLM | Execution Completion Rate (%) | Task Pass Rate (%) | #Total Tokens |
|------------|-------------|-------------------------------|--------------------|---------------|
| SWE-Agent | GPT-4o | 17.58 | 10.19 | 276807 |
| | Claude 3.5 | 41.67 | 22.23 | 456284 |
| | DeepSeek V3 | 18.52 | 12.04 | 414302 |
| OpenHands | GPT-4o | 21.30 | 14.82 | 764521 |
| | Claude 3.5 | 53.70 | 40.74 | 2882928 |
| | DeepSeek V3 | 45.37 | 26.85 | 4749735 |
| RepoMaster | GPT-4o | 48.14 | 40.74 | 250245 |
| | Claude 3.5 | 75.92 | 62.96 | 154913 |
| | DeepSeek V3 | 61.11 | 44.44 | 255998 |

set and empirically tuned k to maximize the recall of important file sets. We gradually increased k values and measured recall rates. As shown in Table 6, the recall rate increases with k , but the marginal gain diminishes significantly beyond $k = 20$. While top-30 achieves slightly higher recall (73.3%), the 3.3% improvement over top-20 (70.0%) does not justify the additional context consumption. Therefore, we selected $k = 20$ as it balances recall performance with context efficiency.

Table 6: Effect of top-k module selection on important file recall rate.

| Top-k | Recall Rate (%) | Relative Improvement |
|--------|-----------------|----------------------|
| Top-5 | 46.6 | – |
| Top-10 | 60.0 | +13.4% |
| Top-20 | 70.0 | +10.0% |
| Top-30 | 73.3 | +3.3% |

The top-10 class selection threshold was determined based on the average length of each class and the total token budget allocated for classes in the initial context (approximately 8k tokens per interaction).

G.4.2 Weight Sensitivity Analysis for Importance Scoring To validate our equal-weighting strategy ($w_i = 1$ in Equation 2), we conducted sensitivity experiments by varying the weights of the six features. We selected 5 repositories with 30 manually identified core files, tuned different weight combinations to optimize important file recall, and evaluated 10 corresponding end-to-end tasks using DeepSeek V3.

Table 7 presents the experimental results. The first row shows the baseline with equal weights, achieving 70% recall, 60% execution completion, and 30% task pass rate. Rows 2-7 show ablation results where each feature’s weight is increased to 5 while others remain at 1. The last three rows show optimized weight combinations that maximize recall.

Key observations:

- **Dependency, Usage, and Git features** show greater impact on recall rate for important files compared to other features. Increasing their weights (rows 2, 4, 7) improves recall by 6.67-10% over baseline.
- **End-to-end task completion quality is less sensitive to weight variations.** Despite recall differences of up to 20% (60% to 80%), task pass rates remain stable at 30-40%.
- **Execution time bottleneck is not in file identification.** End-to-end execution of each task takes approximately 20 minutes, compared to only ~ 4.8 seconds to retrieve important files for typical repositories (1-3k files). This indicates that while entry-point file identification provides a useful starting point, the effectiveness of subsequent exploration and localization strategies is far more critical for overall performance.

These findings validate our choice of equal weighting: (1) it offers competitive performance with minimal complexity, (2) initial file recall has limited impact on final task success, and (3) the exploration phase, not the initialization phase, is the primary determinant of end-to-end performance.

Table 7: Sensitivity analysis of feature weights in importance scoring. Equal weights (baseline) vs. varied weights.

| Feature Weights | | | | | | Performance Metrics | | |
|-----------------|--------|-------|------|-----|-----|---------------------|-----------------|---------------|
| Dep. | Cmplx. | Usage | Sem. | Doc | Git | Recall (%) | Exec. Comp. (%) | Task Pass (%) |
| 1 | 1 | 1 | 1 | 1 | 1 | 70.00 | 60 | 30 |
| 5 | 1 | 1 | 1 | 1 | 1 | 76.67 | 60 | 30 |
| 1 | 5 | 1 | 1 | 1 | 1 | 66.67 | 50 | 30 |
| 1 | 1 | 5 | 1 | 1 | 1 | 73.33 | 70 | 30 |
| 1 | 1 | 1 | 5 | 1 | 1 | 63.33 | 50 | 30 |
| 1 | 1 | 1 | 1 | 5 | 1 | 60.00 | 60 | 30 |
| 1 | 1 | 1 | 1 | 1 | 5 | 76.67 | 60 | 30 |
| 6 | 2 | 4 | 1 | 1 | 6 | 76.67 | 70 | 40 |
| 6 | 2 | 4 | 1 | 1 | 8 | 80.00 | 70 | 30 |
| 6 | 2 | 4 | 1 | 1 | 10 | 76.67 | 60 | 30 |

Table 8: 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 |
| chai-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 | FALSE | 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 | FALSE | 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 | FALSE | 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 9: 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 |
| chai-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 | FALSE | 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 | FALSE | 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 | FALSE | 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 | FALSE | TRUE |
| tgs-salt-identification-challenge | 0.52210 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |

Table 10: 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 | FALSE | TRUE |
| chaii-hindi-and-tamil-question-answering | 0.31169 | 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 | 0.84673 | 0.00050 | 0.00539 | 0.04598 | 0.47205 | FALSE | FALSE | FALSE | FALSE | FALSE | 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 | FALSE | 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 | FALSE | TRUE |
| random-acts-of-pizza | 0.59387 | 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 | 0.62857 | 0.16506 | 0.26996 | 0.29381 | 0.41879 | FALSE | FALSE | FALSE | FALSE | FALSE | 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 | FALSE | TRUE |
| tgs-salt-identification-challenge | 0.50440 | 0.89097 | 0.87163 | 0.85918 | 0.81665 | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |

Table 11: 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 |
| chaii-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 12: 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 | FALSE | TRUE |
| aptos2019-blindness-detection | - | 0.93051 | 0.91965 | 0.91449 | 0.88891 | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| chaii-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 |