

On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents

Jai Lal Lulla

Singapore Management University
Singapore, Singapore
jailal.l.2025@phdcs.smu.edu.sg

Jie M. Zhang

King's College London
London, United Kingdom
jie.zhang@kcl.ac.uk

Seyedmoein Mohsenimofidi

Heidelberg University
Heidelberg, Germany
s.mohsenimofidi@uni-heidelberg.de

Matthias Galster

University of Bamberg
Bamberg, Germany
mgalster@ieee.org

Sebastian Baltes

Heidelberg University
Heidelberg, Germany
sebastian.baltes@uni-heidelberg.de

Christoph Treude

Singapore Management University
Singapore, Singapore
ctreude@smu.edu.sg

Abstract

AI coding agents such as Codex and Claude Code are increasingly used to autonomously contribute to software repositories. However, little is known about how repository-level configuration artifacts affect operational efficiency of the agents. In this paper, we study the impact of AGENTS.md files on the runtime and token consumption of AI coding agents operating on GitHub pull requests. We analyze 10 repositories and 124 pull requests, executing agents under two conditions: with and without an AGENTS.md file. We measure wall-clock execution time and token usage during agent execution. Our results show that the presence of AGENTS.md is associated with a lower median runtime ($\Delta 28.64\%$) and reduced output token consumption ($\Delta 16.58\%$), while maintaining a comparable task completion behavior. Based on these results, we discuss immediate implications for the configuration and deployment of AI coding agents in practice, and outline a broader research agenda on the role of repository-level instructions in shaping the behavior, efficiency, and integration of AI coding agents in software development workflows.

CCS Concepts

- Software and its engineering → Software configuration management and version control systems.

Keywords

AI coding agents, AGENTS.md, pull requests, efficiency

ACM Reference Format:

Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude. 2026. On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents. In *Proceedings of Journal Ahead Workshop (JAWS) 2026 (ICSE JAWS 2026)*. ACM, New York, NY, USA, 5 pages. <https://doi.org/XXXXXXX.XXXXXXX>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE JAWS 2026, Rio de Janeiro, Brazil

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
<https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

AI-assisted software development has evolved rapidly from tools that support individual programming actions to systems that can autonomously carry out multi-step development tasks. Large language models are now routinely used for code generation, testing, repair, review, and documentation, covering substantial portions of the software development lifecycle [6, 10, 11]. Building on these capabilities, recent AI coding agents, such as OpenAI Codex and Claude Code, can navigate repositories, reason over multiple files, execute commands, and submit pull requests with limited human intervention, effectively acting as autonomous contributors rather than passive assistants [17, 20].

As these agents operate with increasing autonomy, their behavior depends not only on model capabilities but also on the contextual information provided by the repository. Previous research has largely focused on interaction-level prompt engineering and agent planning strategies [18], while less attention has been paid to persistent, repository-level artifacts that encode project-specific knowledge. In practice, developers have begun to introduce agent context files such as AGENTS.md or CLAUDE.md that serve as “READMEs for agents,” specifying architecture, build commands, coding conventions, and operational constraints [15]. The AGENTS.md format, for example, has been adopted by more than 60,000 repositories to date [1]. These files shift agent guidance from ephemeral prompts toward version-controlled, inspectable, and collaboratively maintained configuration artifacts.

Recent empirical studies have established agent context files as a widespread and evolving phenomenon in open-source software. Chatlathanagulchai et al. show that such files are actively maintained, structurally consistent, and heavily focused on functional instructions such as build, testing, and implementation details, while non-functional concerns such as performance and security are comparatively rare [5]. Similarly, Mohsenimofidi et al. document the emergence of agent context files as a mechanism for context engineering in agentic coding workflows, highlighting their role in shaping agent behavior across repositories [15]. Together, these studies establish agent context files as a new class of software artifacts that play a central role in agentic development.

Although prior work characterizes the structure, content, and evolution of agent context files, their concrete impact on the behavior of AI coding agents remains largely unexplored. In particular,

there is little empirical evidence on how repository-level instructions influence the behavior of AI coding agents once they are deployed in real development settings. As agents become more deeply integrated into continuous development workflows, understanding their behavioral and operational implications becomes increasingly important, with potential consequences for cost, scalability, and workflow integration.

In this paper, we take a first step toward understanding the impact of AGENTS.md files on AI coding agents by conducting an empirical study on real GitHub pull requests. Inspired by recent work on cost-efficient software engineering agents [8] and sustainable software engineering [19], we compare agent executions with and without an AGENTS.md file, focusing on the operational efficiency of agents as a concrete and measurable aspect of agent behavior. Specifically, we analyze wall-clock execution time and token consumption, which directly relate to agent latency and computational cost in practice. Beyond this focused investigation, we outline a broader research roadmap that situates agent efficiency as one dimension within a more comprehensive study of how AGENTS.md files influence the behavior and integration of AI coding agents in software development workflows.

2 Background and Related Work

Repository-maintained instruction files are increasingly being adopted as a mechanism to align AI coding agents with project-specific norms and expectations. In particular, the use of a repository-level AGENTS.md file has emerged in multiple agent tooling ecosystems. OpenAI Codex documents layered instruction discovery that incorporates repository-level AGENTS.md files [16], while GitLab Duo similarly describes project-level AGENTS.md files to scope and guide agent behavior [7]. Together, these efforts suggest that persistent, developer-curated instruction artifacts are becoming a practical and shared interface for shaping agent behavior in real software repositories.

Despite this emerging adoption, we are not aware of empirical studies that isolate the impact of adding an AGENTS.md file on AI coding agent behavior or efficiency. In particular, no prior work has evaluated how such instruction files affect token usage or wall-clock completion time under a paired, same-task/same-repository evaluation design. This gap motivates the present study.

Previous work has examined the performance and efficiency of LLM-based autonomous coding agents in repository-level tasks. Benchmarks such as *SWE-bench* operationalize this setting using real GitHub issues paired with executable repository snapshots, enabling controlled comparisons of autonomous agents based on issue resolution success and test pass rates [13]. Building on this benchmark, recent agentic systems demonstrate that agent–repository and agent–tool interface design can significantly affect both performance and resource usage. For example, SWE-agent emphasizes agent–computer interface design and context management for repository navigation, editing, and command execution [20]. *AutoCodeRover* similarly combines LLM reasoning with structured search and debugging signals to reduce cost and improve effectiveness on SWE-bench variants [21]. These works motivate measuring not only task success, but also practical deployment costs such as wall-clock completion time and token usage.

Recent work has also begun to characterize developer-provided, persistent context artifacts for AI coding agents, including files such as `copilot-instructions.md` and `CLAUDE.md` [5, 12, 15]. However, we are not aware of studies that *isolate* the effect of introducing an AGENTS.md file on agent efficiency, holding tasks, repositories, and agent architecture constant.

3 Study Design

This study investigates how the presence of an AGENTS.md file affects the performance of autonomous AI coding agents when executing development tasks. We model a standard developer workflow: an agent receives a task description akin to a GitHub issue (“ticket”), performs repository-local actions (e.g., bug fixes, feature additions, refactors), and produces a code change intended to match an expected outcome.

We focus on the *efficiency* of AI coding agents, defined as the computational resources required to complete a development task. This leads to the following research question:

RQ: Does the presence of an AGENTS.md file in the repository root reduce the resources required by an autonomous AI coding agent to complete a development task?

We operationalize “resources” using (i) token usage and (ii) wall-clock time-to-completion measured during each task run. We compare these metrics between paired runs performed with and without an AGENTS.md file in the repository root, keeping the task and repository snapshot constant.

3.1 Data Collection and Analysis

3.1.1 Agent Selection. The agent used in this study is *OpenAI Codex*. At the time of experimentation, the latest available Codex model was `gpt-5.2-codex` [2], which we use consistently across all experiments. We selected Codex because it is specifically designed for software engineering tasks, supports repository-scale context and tool use, and is representative of production-grade AI coding agents used in practice. Moreover, AGENTS.md was first used with Codex before becoming an open format. Evaluating the effect of AGENTS.md across different agent systems and model families is an important direction for future work and part of our ongoing research agenda (Section 5).

We implemented a lightweight Python wrapper to interface with the Codex CLI and to automate task setup and metric collection, available in our online appendix [14]. In the following, the term *agent* refers to this Codex-based agent configuration.

3.1.2 Repository Sampling and Inclusion Criteria. We begin from a corpus of repositories sampled in previous work [15] that analyzed the adoption of agent instruction files, such as AGENTS.md. In that corpus, repositories may contain (i) multiple instruction files of different names, (ii) multiple copies in different subdirectories, or (iii) only one file at the root. In this study, we focus on the simplest configuration: repositories that contain one AGENTS.md file only at the repository root. This configuration minimizes confounding effects from overlapping or conflicting instruction files and enables clearer attribution of observed agent behavior to a single, well-defined source of repository-level guidance. Applying this constraint yields 89 repositories (from 132 total).

To ensure that our evaluation focuses on AGENTS.md files that plausibly provide actionable project context to an AI coding agent, we further restrict the dataset based on the content of the instruction files. Concretely, we filter the root AGENTS.md files using content categories derived from the taxonomy developed in [15]. This taxonomy characterizes AGENTS.md content along dimensions such as coding conventions and best practices, architecture and project structure, project description, testing instructions, and security. We retain only those files that contain information related to (i) conventions and best practices, (ii) architecture and project structure, and (iii) project description. These categories were selected because they are the most common ones [15] and they capture core project knowledge that developers typically need in order to understand and contribute to a codebase [9].

The classification of AGENTS.md files was performed using an LLM (gpt-oss-120b) according to the above criteria. The model was run using Ollama [3], followed by manual verification of the filtered results. After applying this filtering step, we retain 26 repositories, each containing a qualifying root AGENTS.md file.

3.1.3 Pull Request Selection and Task Construction. From the 26 repositories, we randomly sample 10 and select up to 15 merged pull requests (PRs) from each. The cap reflects resource constraints while still enabling coverage across repositories. To obtain task instances that can be executed repeatedly and comparable across repositories, we restrict PRs to small-scope, code-changing contributions and ensure that each PR post-dates the introduction of AGENTS.md in the repository. Each PR satisfies the following criteria: (1) Size constraint: total additions + deletions ≤ 100 LoC; (2) Scope constraint: ≤ 5 modified files; (3) Status: merged PRs only; (4) Temporal constraint: PR created and merged after the introduction of AGENTS.md in that repository; (5) Change type constraint: PR modifies code files only (excluding documentation and configuration changes).

The size and scope constraints reduce the variance from large refactorings and keep agent runs tractable and repeatable. The change-type constraint avoids skew from PRs that do not reflect code changes (e.g., documentation-only updates or version bumps). Together, these restrictions help isolate the effect of AGENTS.md in this initial study. Relaxing them to include larger changes and a broader range of PR types is part of our longer-term research agenda (Section 5).

3.1.4 Reconstructing Pre-PR Repository State. To recreate realistic development conditions, we reconstruct each repository to the state immediately before the selected PR was merged. The agent is then tasked with recreating the PR’s changes from this pre-merge state. Concretely, for each PR we:

- (1) Check out the repository at the pre-merge commit.
- (2) Extract the AGENTS.md version that existed at that commit.
- (3) Run the agent on the pre-merge repository snapshot.

This produces an evaluation setup grounded in an actual, historically merged change, where the repository contents and instruction file (when present) exactly match what a developer or agent would have observed at that point in time.

3.1.5 Issue Generation for PRs Lacking Usable Descriptions. Many PRs do not contain sufficient natural-language context (e.g., empty

body, “fix bug” titles, missing linked issues). To provide consistent and informative task input, we generate a GitHub-issue-style task statement for each PR using a local LLM (gpt-oss-120b). The model is prompted with: (1) the PR diff (patch) and (2) the repository structure at pre-merge state (e.g., file tree).

The output is a structured task description that resembles a GitHub issue (problem statement, expected behavior, constraints, and acceptance criteria). This step standardizes the agent’s input format across PRs and reduces variance introduced by incomplete PR metadata [4]. The generated task descriptions are available online [14].

3.1.6 Experimental Conditions. We run the agent on each task instance under two conditions:

- **With AGENTS.md:** The repository snapshot includes the extracted root AGENTS.md from the pre-merge commit.
- **Without AGENTS.md:** The same snapshot is used, but the AGENTS.md file is removed (all other files unchanged).

In both conditions, the agent is provided with the same task input, namely the GitHub-like issue generated earlier for each pull request. The agent then produces code changes corresponding to the original PR. The resulting setup constitutes a **paired within-task design** that controls for repository, task, and codebase state, while varying only the presence of AGENTS.md.

3.1.7 Running the Experiments. To minimize confounding factors and ensure repeatability, experiments were conducted in isolated Docker environments at the repository level. For each repository, we instantiated a fresh container and cloned the target repository into a temporary working directory. Each task instance was executed by checking out the corresponding pre-merge commit within the container. After task completion, the repository was reset to a clean state (discarding all local changes), and the workspace was cleaned before checking out the commit associated with the next task. The code used to run the experiments, including Dockerfiles, is provided as part of the supplementary material [14].

The agent was granted access only to this sandboxed environment and could modify files exclusively within the container. No state (e.g., caches, artifacts, or intermediate files) was reused across tasks beyond the version-controlled repository contents, ensuring that every task began from an identical repository snapshot.

3.1.8 Metrics. We collected the following operational performance metrics during each run:

- **Token usage:** Total tokens consumed, comprising input tokens, cached input tokens, and output tokens.
- **Time-to-completion:** Wall-clock time for the agent to produce its final output (in seconds).

A comprehensive evaluation of the output quality, e.g., the semantic correctness or the functional equivalence to the merged PR, is beyond the scope of this paper. However, it is part of our research roadmap described in Section 5. Nevertheless, to ensure that the observed efficiency differences are not simply due to agents producing degenerate or trivially incomplete output, we performed a manual sanity check. Specifically, we randomly sampled 50 PR tasks and inspected the corresponding agent outputs, comparing them against the human-written merged pull requests, to confirm that they resulted in non-empty, non-trivial code changes consistent with the

Table 1: Resource Usage With and Without AGENTS .md

Metric	Without	With	Diff	Δ%
Wall-Clock Time (s)*				
Mean	162.94	129.91	33.03	20.27%
Median	98.57	70.34	28.23	28.64%
Std Dev	182.24	136.84	45.40	24.91%
Input Tokens				
Mean	353,010.01	318,651.51	34,358.50	9.73%
Median	116,609.00	120,587.00	-3,978.00	-3.41%
Std Dev	654,603.95	510,776.51	143,827.43	21.97%
Cached Input Tokens				
Mean	328,877.31	296,078.73	32,798.58	9.97%
Median	103,424.00	104,448.00	-1,024.00	-0.99%
Std Dev	632,622.27	494,157.89	138,464.38	21.89%
Output Tokens*				
Mean	5,744.81	4,591.46	1,153.35	20.08%
Median	2,925.00	2,440.00	485.00	16.58%
Std Dev	6,987.74	5,161.67	1,826.06	26.13%

* Statistically significant difference, Wilcoxon signed-rank test ($p < 0.05$).

intended task, rather than aborted runs or random edits. Although this sanity check does not constitute a full correctness evaluation, it provides confidence that the efficiency measurements reported in this paper are not driven by obvious failures or reductions in task execution.

4 Results

Token usage. Providing an AGENTS .md file reduces generation cost. Mean output token usage decreases from 5,744.81 tokens (without AGENTS .md) to 4,591.46 tokens (with AGENTS .md), a reduction of 1,153.35 tokens ($\approx 20.08\%$). Median output tokens decrease more modestly, from 2,925.00 to 2,440.00 (485 tokens, $\approx 16.58\%$). Mean input and cached input tokens also decline slightly (353,010.01 to 318,651.51; 9.73% and 328,877.31 to 296,078.73; 9.97%, respectively), with medians essentially unchanged or slightly higher. The larger reduction in mean output tokens compared to the median suggests that AGENTS .md primarily reduces token usage in a small number of very high-cost runs, rather than uniformly lowering token consumption across all task instances.

Wall-clock time-to-completion. Agents provided with an AGENTS .md file complete tasks faster than agents operating without it. Mean wall-clock time-to-completion decreases from 162.94s (without AGENTS .md, std dev 182.24s) to 129.91s (with AGENTS .md, std dev 136.84s), an absolute reduction of 33.03s ($\approx 20.27\%$). Median completion time shows a similar reduction, decreasing from 98.57s to 70.34s (28.23s, $\approx 28.64\%$). The close alignment between mean and median improvements indicates that the reduction is not driven solely by a small number of extreme runs, but reflects a general shift toward faster task completion.

5 Research Roadmap

This study represents an initial step towards understanding how AGENTS .md files influence the behavior of autonomous AI coding agents. Our results provide early evidence that such files can affect agent efficiency in realistic development tasks. At the same time, the study focuses on a sampled subset of repositories, pull requests, and agent configurations, which naturally motivates the next steps.

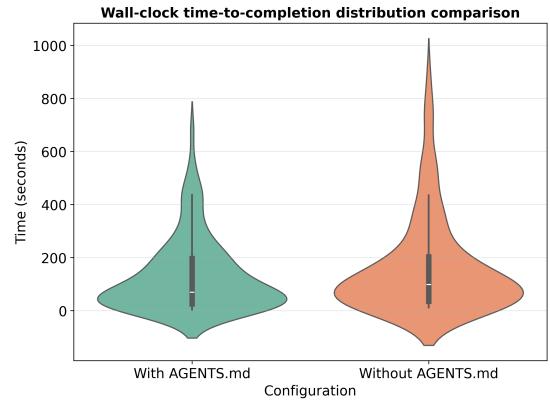


Figure 1: Wall-clock time-to-completion distributions for agent runs with and without AGENTS .md

A first direction for future work is to expand the empirical scope of the study. Although we control for task, repository state, and agent configuration through a paired design, observed effects may still depend on agent stochasticity, the specific agent framework and model used, and the characteristics of the selected tasks. Replicating the study across additional repositories, larger and more diverse pull requests, and multiple agent systems and model families will help assess the robustness and generality of the observed efficiency effects. Similarly, relaxing current constraints on task size and scope will allow us to examine whether the influence of AGENTS .md extends to larger refactorings, multi-module changes, and more complex development scenarios.

A second direction is to investigate dimensions of agent behavior beyond efficiency. In this study, we focus on token usage and wall-clock time as measurable indicators of cost. However, these metrics do not capture whether agent-produced changes are correct, maintainable, or aligned with developer intent. Future work will therefore incorporate correctness and alignment evaluations, for example, by comparing agent-generated changes against the original merged pull requests using automated checks and structural similarity analyzes. Beyond a binary treatment of AGENTS .md as present or absent, we also plan to examine how properties of these files, such as specificity, organization, and the inclusion of workflow guidance, relate to agent outcomes. For example, we speculate that some of the efficiency gains reported in this paper arise because AGENTS .md files describe repository structure and conventions up-front, reducing the need for agents to infer project organization through exploratory navigation. Analyzing agent execution traces will help explain why AGENTS .md files lead to more efficient generation, for example, by fewer planning iterations, reduced exploratory navigation, and fewer repeated requests to the underlying model.

In summary, our results provide initial empirical evidence that repository-level instruction files can have measurable operational effects on autonomous AI coding agents. In particular, we show that the presence of a root AGENTS .md file is associated with reduced token usage and faster task completion on real pull requests. These findings position AGENTS .md as a practical repository-level mechanism for shaping agent behavior and motivate further investigation of its role in agent efficiency, alignment, and integration within software development workflows.

References

- [1] [n. d.]. AGENTS.md. <https://agents.md/>. [Accessed 23-01-2026].
- [2] [n. d.]. Codex Models – developers.openai.com. <https://developers.openai.com/codex/models/>. [Accessed 23-01-2026].
- [3] [n. d.]. Ollama. <https://ollama.com/>. [Accessed 23-01-2026].
- [4] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) (ESEC/FSE 2019). Association for Computing Machinery, New York, NY, USA, 86–96. doi:10.1145/3338906.3338947
- [5] Worawalan Chatlatangulchai, Hao Li, Yutaro Kashiwa, Brittany Reid, Kundjanaith Thonglek, Pattara Leelaprute, Arnon Rungsawang, Bundit Manaskasemsak, Bram Adams, Ahmed E. Hassan, and Hajimu Iida. 2025. Agent READMEs: An Empirical Study of Context Files for Agentic Coding. *arXiv preprint arXiv:2511.12884* (2025).
- [6] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.
- [7] GitLab. 2025. AGENTS.md customization files. GitLab Documentation. https://docs.gitlab.com/user/gitlab_duo/customize_duo/agents_md/ Accessed 2026-01-18.
- [8] Yaoqi Guo, Ying Xiao, Jie M Zhang, Mark Harman, Yiling Lou, Yang Liu, and Zhenpeng Chen. 2026. EET: Experience-Driven Early Termination for Cost-Efficient Software Engineering Agents. *arXiv preprint arXiv:2601.05777* (2026).
- [9] Junda He, Jieke Shi, Terry Yue Zhuo, Christoph Treude, Jiamou Sun, Zhenchang Xing, Xiaoning Du, and David Lo. 2025. LLM-as-a-Judge for Software Engineering: Literature Review, Vision, and the Road Ahead. *arXiv:2510.24367 [cs.SE]* <https://arxiv.org/abs/2510.24367>
- [10] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.
- [11] Ju Young Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2025. A Survey on Large Language Models for Code Generation. *ACM Transactions on Software Engineering and Methodology* (2025).
- [12] Shaokang Jiang and Daye Nam. 2025. An Empirical Study of Developer-Provided Context for AI Coding Assistants in Open-Source Projects. doi:10.48550/arXiv.2512.18925 arXiv:2512.18925
- [13] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv:2310.06770 [cs.CL]* doi:10.48550/arXiv.2310.06770 arXiv:2310.06770v3.
- [14] Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude. 2026. *On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents (Online Appendix)*. doi:10.5281/zenodo.18348507
- [15] Seyedmoein Mohsenimofidi, Matthias Galster, Christoph Treude, and Sebastian Baltes. 2026. Context Engineering for AI Agents in Open-Source Software. In *Proceedings of the 23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026)*.
- [16] OpenAI. 2025. Custom instructions with AGENTS.md. OpenAI Developers Documentation. <https://developers.openai.com/codex/guides/agents-md/> Accessed 2026-01-18.
- [17] Abhik Roychoudhury. 2025. Agentic AI for Software: thoughts from Software Engineering community. *arXiv preprint arXiv:2508.17343* (2025).
- [18] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).
- [19] Colin C Venters, Rafael Capilla, Elisa Yumi Nakagawa, Stefanie Betz, Birgit Penzenstadler, Tom Crick, and Ian Brooks. 2023. Sustainable software engineering: Reflections on advances in research and practice. *Information and Software Technology* 164 (2023), 107316.
- [20] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems* 37 (2024), 50528–50652.
- [21] Yunlong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24)*. doi:10.1145/3650212.3680384 Also available as arXiv:2404.05427.

Received 23 January 2026