Repository-Local Decision Context for Coding Agents: An Exploratory Pilot

Anonymous authors
Paper under double-blind review

Abstract

Large language model agents can generate plausible patches quickly, but production changes must also preserve product intent and architectural decisions. We study repository-local decision context: versioned product constraints, architecture decisions, task guidance, review policy, and escalation boundaries. An eight-task pilot planned 96 one-model runs across search-only, global-document, task-local, and full review-and-revision conditions. A validity audit quarantined seven initial outcomes and restarted execution under corrected specification 2026-09-06.2. The corrected run stopped after 32 outcomes with interim results visible. Observed hidden-test success was Search-only 2/5 (40.0%), Global docs 6/11 (54.5%), Local-Context 7/11 (63.6%), and FullWorkflow 3/5 (60.0%). Thus local context exceeded search-only by 23.6 percentage points in the aggregate sample. Among the four tasks represented in both conditions, local context tied search-only on three and improved on one, for a task-macro difference of +25.0 points; removing the improved task reduces that difference to zero. Full workflow and local context tied on all three shared tasks. These 32 runs provide positive but task-concentrated evidence for repository-local context and no observed incremental benefit from agent review. Unequal cells, task clustering, and outcome-visible stopping prevent a confirmatory population treatment claim.

1 Introduction

Coding agents can transform a natural-language task into a plausible software change, but correctness in production repositories is broader than passing visible tests. A change can satisfy a local specification while violating an architecture decision, mishandling an unstated product edge case, or taking an action that should have required human approval. SWE-bench (Jimenez et al., 2024) measures whether agents can complete issue-resolution tasks. We ask a different question: which human decisions must be explicit for an agent to produce a reliable, reviewable change?

We define repository-local decision context as versioned information stored with, or resolvable from, the codebase: product intent, behavioral edge cases, architecture decision records, task-local implementation constraints, review policies, and escalation boundaries. The pilot compares four bundled access/workflow conditions under fixed model, tool, and execution limits; it does not identify every component separately.

This paper contributes:

1. an auditable eight-task pilot whose evaluators combine hidden behavior with separate machine-checkable architecture, security, and scope rules;
2. a model-agnostic representation of task-local product and decision context and a four-condition execution protocol;
3. a validity record documenting how evaluator–prompt mismatch invalidated the initial outcomes in this pilot and how those outcomes were quarantined;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

4. aggregate and task-overlap evidence from 32 corrected runs, including a leave-one-task-out sensitivity check; and
5. an industrial case that motivates the benchmark without being used as causal evidence of productivity.

2 Research Questions and Hypotheses

RQ1: Reliability. Does task-local decision context improve hidden-test success and architectural conformance relative to search-only access or global documentation?

RQ2: Efficiency. How do context interventions affect wall-clock time, tool use, and unnecessary scope changes?

RQ3: Review and revision. In the observed full-workflow runs, does independent agent review and bounded revision coincide with better final patches, and what time does it add?

RQ4: Context failure. How sensitive are agents to missing, contradictory, or stale decision context?

The original protocol specified directional hypotheses: task-local context would reduce architecture violations, and independent review would catch some functional and security failures. The early-terminated pilot is not a valid confirmatory test of those claims. We use the research questions to organize exploratory summaries and treat context staleness and contradiction as future work because those ablations were not executed.

3 Related Work

Repository-level coding benchmarks. SWE-bench evaluates whether models can resolve real issues in public repositories, moving code generation beyond isolated functions to changes that must satisfy repository tests (Jimenez et al., 2024). CrossCodeEval studies cross-file completion and shows why repository dependencies matter even before an agent begins editing (Ding et al., 2023). These benchmarks establish repository-scale evaluation, but functional tests alone do not capture whether a patch respects design rationale, scope boundaries, or product decisions.

Retrieval and agent–computer interfaces. RepoCoder alternates retrieval and generation to surface useful cross-file context (Zhang et al., 2023). SWE-agent argues that the interface between a language model and a development environment materially affects software-task performance (Yang et al., 2024), while OpenHands provides an open platform for generalist software agents and reproducible agent implementations (Wang et al., 2025). Agentless demonstrates that a comparatively simple localize–repair workflow can be competitive with elaborate autonomous systems (Xia et al., 2025). These studies show why comparisons must control tools, interfaces, and context budgets before attributing a difference to the model.

Feedback, review, and refinement. Reflexion uses verbal feedback retained across attempts to improve agent behavior (Shinn et al., 2023); Self-Refine iteratively asks a model to critique and revise its own output (Madaan et al., 2023). Our independent reviewer receives a frozen rubric and cannot silently alter the task. Coder–reviewer exchanges stop after a fixed bounce limit or escalate to a human. The ablations test whether review adds information beyond another generation pass.

Architecture decisions and human productivity. Architecture decision records make a decision, its context, alternatives, and consequences explicit (Tyree & Akerman, 2005). We encode this practice as machine-visible treatment material and test it against repository search alone. A controlled study found that an AI pair programmer reduced completion

Table 1: Pilot conditions. Tool permissions, model settings, and resource budgets were held constant unless they were the factor under study.

Condition	Information and workflow
Search-only	Repository search and inspection tools; no authored decision-context artifacts.
Global documentation	Search tools plus repository-wide product and architecture documentation.
LocalContext	Search tools plus task-local product, architecture, and validation context.
FullWorkflow	LocalContext plus independent agent review, bounded coder–reviewer iterations, and human escalation rules.

time on a bounded programming task (Peng et al., 2023). That result supports our human-effort measure. It does not validate observational productivity comparisons for long-running team projects.

Gap addressed. Prior work separately studies issue resolution, repository retrieval, agent interfaces, and iterative refinement. Our pilot compares retrieved repository text with explicit human decisions about acceptable behavior and architecture while holding the model, tools, and execution limits fixed. Context volume was not token-matched, so representation and quantity remain confounded.

4 Repository-Local Decision Context

4.1 Context representation

Each benchmark task may expose one or more of the following versioned artifacts:

- Product intent: users, desired outcomes, invariants, and explicitly enumerated edge cases;
- Architecture decisions: selected option, rejected alternatives, rationale, constraints, and consequences;
- Task-local guidance: affected modules, interfaces, dependencies, validation commands, and definitions of done;
- Review policy: functional, architecture, security, and scope checks applied consistently during implementation and review; and
- Escalation policy: ambiguity and risk conditions under which the agent must ask, abstain, or transfer control to a human.

The representation is model-agnostic and stored in plain text or structured files. Every treatment records exactly which artifacts and repository tools are visible to the agent.

4.2 Experimental conditions

Table 1 defines the four primary conditions. The first three isolate how decision context is represented; FullWorkflow adds independent review and bounded escalation to LocalContext. Each condition begins from the same clean worktree and issue request.

5 Benchmark Design

5.1 Repositories and tasks

The pilot used eight tasks with executable tests and machine-checkable constraints. Four came from public Python repositories and four were recreated from historical changes in two internal TypeScript services. Each task included:

1. an issue-style request visible to all conditions;
2. hidden behavioral tests unavailable to the agent;
3. deterministic architecture, security/privacy, or scope rules;
4. reference evidence documenting intended behavior; and
5. provenance and treatment-artifact hashes.

The planned design crossed every task with four context conditions and three repetitions under one model. Fresh worktrees prevented information transfer. Because execution stopped after the first 32 randomized assignments, the observed data are not a complete cross: condition and task counts differ, and some task–condition cells are empty. This imbalance is reported rather than repaired by post hoc outcome selection.

We constructed an eight-task pilot containing four public issue-resolution tasks from Django, scikit-learn, Sphinx, and xarray and four historical tasks from two internal services. Each task had one assignment in every condition for each of three repetitions, yielding a planned one-model total of 96 runs. The run order was randomized once and frozen. Public tasks used canonical issue statements; internal prompts were rewritten to expose every behavior observable by the functional evaluator. Internal repositories, tests, and traces remain private pending redistribution review.

The pilot was intended to validate execution and estimate variance before a separate confirmatory study. It was not powered as that study. A pre-outcome simulation under the planned repeated-measures model showed that even all 96 pilot runs would provide limited power for small effects. The author later directed termination after 32 corrected outcomes, with interim outcomes already visible. We therefore analyze the observed randomized prefix descriptively and do not treat run count as an independent sample size or report a confirmatory treatment effect.

5.2 Models, tools, and budgets

The executed pilot used one frozen model selector, gpt-5.4-2026-03-05, through the same private agent-service interface for every condition. Attempts to validate a second model family failed before outcome collection, and the resulting one-model scope was recorded as a pre-outcome deviation. Service defaults did not expose temperature or seed. Every run received the same shell, editor, repository-search, and test tools and was limited to 200,000 input tokens, 40,000 output tokens, 125 tool calls, and 90 wall-clock minutes. The full-workflow condition additionally invoked the frozen independent-review and bounded-revision prompts.

Trusted provider token and billing receipts were unavailable through this interface. Following the frozen accounting policy, we report wall-clock time and trace-derived tool use where available, but do not impute token counts or commercial cost. The 32-run pilot did not execute the planned token-matched or model-interaction sensitivity analyses.

Cost accounting. Because neither the actual routed-model receipt nor validated provider token counts were available, commercial-equivalent cost is unavailable. Character counts are retained only as diagnostics and are not converted into billing claims.

5.3 Outcome measures

Primary outcome. The corrected primary outcome is binary functional hidden-test success. A run passes only when all evaluator-defined behavioral checks pass. Timeouts, invalid patches, and test failures remain functional failures.

Secondary outcomes. Deterministic rules separately count behavior, architecture, security/privacy, and scope violations. We also retain wall-clock time, patch size, changed files, trace-derived tool calls, review iterations, and infrastructure disposition. Architecture and security rules do not alter the primary functional outcome. Human correction time, token use, and cost were not measured reliably and are not reported.

Failure taxonomy. The final descriptive report separates functional failure from behavior, architecture, security/privacy, and scope-rule violations. Infrastructure failures are missing outcomes rather than treatment failures and are listed separately.

6 Experimental Protocol

6.1 Assignment and execution

A frozen 96-row plan randomized the crossed task–condition repetitions. Every run used a fresh worktree and an isolated agent-service process. Sandboxing and prompt construction prevented agents from reading hidden tests, reference patches, evaluator code, or treatment artifacts assigned to another condition. The runner atomically checkpointed each completed summary, evaluation, patch, and trace; interrupted runs were not counted as outcomes.

6.2 Automated custodian evaluation

A separate custodian applied hidden functional tests and deterministic behavior, architecture, security/privacy, and scope rules after patch generation. Before v2 execution, all eight evaluators were tested against an unchanged base, a reference repair, and a deliberately faulty repair: bases and faulty repairs had to fail, while reference repairs had to pass. Evaluator artifacts were hashed and hidden-check isolation was verified in fresh worktrees. These checks score patches rather than model explanations.

The full-workflow treatment used an independent agent reviewer with a frozen rubric and at most two coder–reviewer revision cycles. This is a treatment component, not human outcome assessment. No blinded human correction-time study was run, so the paper makes no claims about reviewer agreement or human effort.

6.3 Specification repair and stopping

After seven outcomes under the initial specification, an audit found that some task prompts omitted evaluator-observable behavior and that functional success had been conflated with secondary architecture and security rules. All seven outcomes were quarantined, prompts and scoring were corrected, new run IDs were minted, and execution restarted as specification 2026-09-06.2 while preserving assignment order. No v1 outcome enters the analysis.

The corrected plan still contained 96 runs. After 26 corrected outcomes were visible, the author directed the experiment to stop at 32 and requested final analysis. A process guard stopped execution after the 32nd durable checkpoint; any started but incomplete next run is excluded. This outcome-visible optional stopping is a material deviation and precludes the preregistered confirmatory analysis.

6.4 Descriptive analysis

We report counts, raw hidden-test success proportions, Wilson 95% binomial intervals within each observed condition, deterministic violation totals, and mean and median wall-clock time. Raw risk differences versus search-only describe the observed sample. We also compare conditions on tasks represented in both cells, average those within-task rate differences, and remove each overlapping task in turn to expose concentration of the signal. These sensitivity summaries reduce, but do not eliminate, composition differences because repetitions remain unequal. We do not report p-values, odds ratios, or a fitted mixed-effects treatment estimate: the prefix is unbalanced, repeated runs are clustered, and the stop decision followed outcome access. The summaries identify patterns in this pilot sample rather than a population treatment effect.

6.5 Contamination and leakage controls

Public issue requests may have appeared in model training data; internal tasks reduce but do not eliminate that concern. Agents could inspect only their clean repository snapshot and

Table 2: Corrected-v2 pilot outcomes. Conditions contain unequal task mixtures. Wilson intervals describe each observed cell; violation columns are raw deterministic rule-failure counts, separate from functional success.

Condition	n	Hidden-test pass (95% CI)	Behavior	Architecture	Security/privacy	Scope	Mean time
Search-only	5	2/5 (40.0%; 11.8%–76.9%)	3	1	0	3	241.1s
Global docs	11	6/11 (54.5%; 28.0%–78.7%)	9	6	0	1	236.4s
LocalContext	11	7/11 (63.6%; 35.4%–84.8%)	4	4	0	3	85.2s
FullWorkflow	5	3/5 (60.0%; 23.1%–88.2%)	2	1	0	2	342.5s

assigned context. Hidden tests, reference solutions, and other conditions' artifacts remained outside the worktree and were applied by the custodian after generation. We do not claim that the pilot isolates training exposure from treatment effects.

6.6 Deviation reporting

Appendix B records the pre-outcome reduction to one model, the outcome-visible specification repair and quarantine, and the outcome-visible sample-size reduction. It identifies affected outcomes and the analysis response. Timestamped machine-readable details and frozen runner, plan, evaluator-manifest, and treatment hashes accompany the private evidence package.

7 Results

The corrected-v2 pilot produced 32 durable outcomes from 8 tasks, with 18 functional hidden-test passes (56.2%; Wilson 95% CI 39.3%–71.8%). Execution ended with 64 of the planned 96 assignments unrun or incomplete. Seven earlier v1 outcomes are quarantined and absent from every number below (Section 6.3).

Aggregate functional outcome. Observed raw condition rates were Search-only 2/5 (40.0%), Global docs 6/11 (54.5%), LocalContext 7/11 (63.6%), and FullWorkflow 3/5 (60.0%). Every context-bearing condition exceeded search-only in this sample. The raw local-context minus search-only difference was +23.6 percentage points; the full-workflow difference was +20.0 points and the global-document difference was +14.5 points. These differences describe the observed runs. Because task mixtures differ and several task–condition cells are empty, they are not population treatment effects.

Task-overlap sensitivity. Four tasks had outcomes under both search-only and local context. Local context tied search-only on three and improved on one; it was worse on none. Averaging the four within-task pass-rate differences gives +25.0 percentage points. The signal is concentrated: leave-one-task-out differences are +33.3 points when any tied task is removed and 0.0 when the improved task is removed. On the five tasks shared by global documents and local context, local tied on four and improved on one (task-macro difference +6.7 points). Full workflow and local context tied on all three shared tasks. These overlap checks reduce task-composition differences but retain unequal repetitions and are descriptive sensitivity analyses.

Secondary checks and time. All evaluator families passed base/reference/fault validation before corrected-v2 execution. Across the 32 reported runs, the custodian recorded 18 behavior, 12 architecture, 0 security/privacy, and 9 scope violations. Mean wall-clock time was 201.7s and the median was 64.5s. Condition means in Table 2 are affected by task mix and bounded review cycles; they are not efficiency estimates.

Interpretation. The 32 corrected runs establish three findings about this pilot sample. First, context-bearing conditions achieved 14.5–23.6 points higher aggregate functional success than search-only. Second, local context never underperformed search-only on common task support, but its gain came from one of four tasks. Third, full workflow showed no

incremental functional advantage over local context on the three tasks they shared. The results support repository-local context as a promising intervention and do not support an incremental review benefit. They do not establish population treatment effects because the prefix is unbalanced, repeated runs are clustered, and stopping followed outcome access.

8 Industrial Motivation

The benchmark is motivated by a longitudinal case at a large enterprise software organization. An early multi-agent prototype produced a running application quickly but accumulated cyclic dependencies, unsuitable architecture, integration assumptions, and pervasive test failures during hardening. A second workflow narrowed execution to one feature, one work item, and one pull request; reviewability improved, but product-edge-case failures remained. A later workflow moved product and architecture judgment before execution, made relevant decisions queryable, separated coder and reviewer roles, bounded review loops, and preserved a human release gate. A contemporaneous presentation depicts a planner decomposing a product requirements document into implementation tasks, separate coder and reviewer agents, and explicit human ownership of product, architecture, and risky decisions.

The source archive contains multiple execution scopes that must not be merged. A curated log enumerates 21 workflow-platform stories and includes configurable per-stage agents, prompts, and review-bounce settings. A separate adoption summary describes more than 40 end-to-end stories over a three-week interval, while the longitudinal production account covers a five-engineer, five-month delivery. We treat these as separate evidence sets until work-item identity, time window, and inclusion rules are reconciled.

An internal matched-product comparison reported approximately five times more output per engineer for the later project. The comparison was observational, bundled several interventions, and was not randomized. It motivates the controlled study but does not support a causal productivity claim.

Case-study evidence boundary. The available archive contains presentation material, execution logs, and later adoption summaries with different units and time windows. We did not complete an item-level reconciliation of the 21-story workflow log, the separate 40-story adoption summary, and the five-month production account. We therefore use the case only to motivate the intervention. The reported $5\times$ comparison is identified as an internal observational estimate and is excluded from benchmark analysis. Product complexity, staffing, prior code, model changes, developer selection, and measurement differences remain uncontrolled explanations.

9 Discussion and Conclusion

The 32 corrected runs answer a narrower question than the planned experiment, but they produce a clear pilot result. Every context-bearing condition had higher aggregate functional success than search-only, and local context was never worse on the four tasks shared with search-only. The aggregate local-context advantage was 23.6 percentage points; the task-macro advantage on shared support was 25.0 points. That advantage came from one task, so it is evidence that local decision context can help rather than evidence that it helps uniformly.

The data do not show that adding an agent reviewer improved functional outcomes: full workflow and local context tied on all three shared tasks. The practical conclusion is therefore specific. Repository-local decision context merits prospective, task-blocked evaluation and deployment behind human review; the incremental value of automated reviewer loops remains unproven. The benchmark's separate functional and policy checks, evaluator isolation, and explicit repair record provide a foundation for that follow-up.

10 Limitations and Threats to Validity

Construct validity. Hidden tests approximate functional correctness but may miss usability, maintainability, and operational behavior. Deterministic rules encode only the selected architecture, security, and scope constraints. A zero violation count therefore does not establish production readiness. We keep functional success and each rule family separate and do not construct a composite quality score.

Internal validity. Context conditions differ in volume, salience, structure, and writing quality as well as decision content. Tools and execution limits were frozen, but the planned token-matched and component-ablation checks were not run. The corrected-v2 evaluator was isolated from agents, yet the repair occurred after seven v1 outcomes had been observed. Those outcomes are excluded, not "corrected."

External validity. Eight tasks, six repositories, and one model-service snapshot cannot represent coding agents generally. Four tasks are public Python issues and four are private TypeScript service changes. Results may change with model, provider, interface, language, repository age, tool availability, or training exposure. The industrial case concerns one organization and is motivation, not validation.

Conclusion validity. Execution stopped after 32 of 96 planned v2 outcomes, following inspection of interim results. The observed prefix is unbalanced by task and condition; some primary-contrast task cells are empty, and repeated runs are clustered. Wilson intervals quantify only within-cell binomial uncertainty and are not adjusted treatment-effect intervals. Raw differences may reflect task composition. Accordingly, we report no p-values or confirmatory mixed-effects estimate, do not interpret null differences as equivalence, and make no causal claim.

Contamination, drift, and case-study validity. Public tasks or reference solutions may have appeared in training data, and model behavior can change across API updates. We retained model and repository provenance but did not have enough completed task cells to repeat analyses after excluding high-exposure tasks. The industrial case covers one organization, bundles several interventions, and cannot establish causal productivity effects.

Context-induced harm. Explicit context can be stale, contradictory, excessively restrictive, or encode organizational bias. The planned staleness and contradiction treatments were not executed, so this pilot does not measure those harms. Human escalation remains necessary for high-impact ambiguity.

AI Use Statement

An OpenAI GPT-family assistant supported source search and summarization, experimental-design critique, benchmark monitoring, analysis-code generation, LaTeX editing, and consistency checks. The evaluated model separately acted as the experimental coding agent under frozen treatment prompts. The human author selected the research question and methods, approved the one-model deviation, directed outcome-visible termination at 32 corrected runs, reviewed the interpretation boundary, and takes responsibility for the paper. Seven pre-repair outcomes and all infrastructure smoke runs are excluded. The final 32-run tables are generated mechanically from corrected-v2 custodian records; the assistant did not override evaluator results or select runs by outcome.

Ethics Statement

The pilot evaluated model-generated patches and reused historical development artifacts; it did not recruit human reviewers or score employees. The author reports internal publication and secondary-use approval from the relevant engineering leadership, but no formal institutional-review identifier was issued. Raw internal prompts, repositories, patches, and

traces remain access-restricted, are separated from employment evaluation, and are subject to a 90-day raw-trace retention policy. Only aggregate, de-identified findings are reported publicly. Public task use remains subject to repository licenses, and external release of internal task material requires separate confidentiality, security, and redistribution review. The observational industrial comparison is not used to rank individuals or claim causal productivity improvement.

Reproducibility Statement

The internal evidence package contains the frozen 96-row plan, treatment and evaluator hashes, corrected task prompts, evaluator validation records, atomic per-run summaries, patches and traces, the complete deviation table, and the script that regenerates the 32-run descriptive tables. The analysis script asserts specification 2026-09-06.2 and refuses to mix the seven quarantined v1 outcomes. Four public task instances and their evaluation assets can support a redistributable subset; four internal tasks cannot be released without additional approval. No clean-room reproduction of an anonymous package has yet been completed. We therefore provide an auditable internal pilot record, not a claim of a fully public reproducibility artifact.

References

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion. In Advances in Neural Information Processing Systems, volume 36, pp. 46701–46723, 2023. doi: 10.52202/075280-2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/920f2dced7d32ab2ba2f1970bc306af6-Abstract-Datasets_and_Benchmarks.html.

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? In International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Advances in Neural Information Processing Systems, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv preprint arXiv:2302.06590, 2023. URL https://arxiv.org/abs/2302.06590.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In Advances in Neural Information Processing Systems, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Jeff Tyree and Art Akerman. Architecture decisions: Demystifying architecture. IEEE Software, 22(2):19–27, 2005. doi: 10.1109/MS.2005.27.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In International

Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. Proceedings of the ACM on Software Engineering, 2 (FSE):854–878, 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In Advances in Neural Information Processing Systems, volume 37, pp. 50528–50652, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 2471–2484. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

A Task Card Schema

Each released task card will make the experimental unit auditable without revealing hidden answers. It records:

- repository URL or redistributable snapshot, revision, language, dependency lock, and license;
- issue statement, task category, affected area, pilot difficulty, and inclusion or exclusion rationale;
- visible setup and tests, hidden-evaluation custodian, architecture and scope checks, and reference-evidence provenance;
- allowed tools, network policy, token and wall-clock budgets, stopping rules, and infrastructure-failure procedure;
- treatment artifact hashes, context-token counts, review and escalation settings, and random assignment identifier; and
- public-history dates, suspected training exposure, transformations, known ambiguities, and disclosure restrictions.

Hidden tests and reference patches will be released only when doing so does not invalidate continued benchmark use; otherwise, a versioned evaluation service and audit procedure will be documented.

B Deviation Record

Table 3: Material deviations. "Visible" indicates whether any relevant outcomes had been inspected when the decision was made.

ID	Timing	Visible	Change	Response and interpretation
D1	Before outcome collection	No	Two models/human scoring → one model/automated custodian	Regenerated plan; scope limited to one snapshot
D2	After 7 v1 outcomes	Yes	Repaired task statements and separated functional from rule outcomes	Quarantined all 7; restarted with new IDs as v2
D3	After 26 v2 outcomes	Yes	Planned 96 v2 runs → stop after 32	Left 64 unrun; descriptive analysis only

D2 outcomes are not corrected or reused, and D3 precludes confirmatory inference. The private deviation ledger records decision timestamps, reasons, affected artifacts, and frozen hashes.

C Automated Evaluation and Workflow Review

The custodian evaluator receives the final patch in an isolated worktree. It runs hidden behavioral tests and deterministic rule checks grouped as behavior, architecture, security/privacy, and scope. Functional success is derived only from the hidden behavioral checks; rule-group counts remain secondary outcomes. For every task, validation required an unchanged base to fail, the reference repair to pass, and a deliberately faulty repair to fail. All 24 validation expectations passed before corrected-v2 execution.

In FullWorkflow, a separate agent reviews the coder's patch against a frozen rubric covering behavior, architecture, security/privacy, scope, and validation. The coder may revise twice. The final custodian, not the reviewing agent, assigns the reported outcomes. No human reviewer score or correction-time estimate is part of this pilot.

D Additional Results and Failure Examples

The machine-readable analysis output reports every completed v2 run's task, condition, functional result, deterministic violation counts, and wall-clock time, together with task-by-condition composition. The companion table reports condition summaries and Wilson intervals. Started but incomplete directories and all 64 unrun assignments are excluded and counted explicitly.

The planned model-family interaction, token-matched sensitivity, context staleness and contradiction ablations, human-review agreement, and confirmatory mixed-effects model were not executed. The deviation table maps each design change to its timing, affected outcomes, and analysis response. We omit post-hoc qualitative "best" and "worst" examples because the planned selection procedure was not completed before outcome access.