PURPAGENT: INTEGRATING SECURITY REASONING INTO CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly automate software development, yet functionally correct implementations can still introduce security vulnerabilities. Integrating security review into these agents requires feedback that supports successful repair within a limited execution budget. In this paper, we introduce PurpAgent Harness, the first scaffold that seamlessly integrates secure coding and security review. Its key idea is to translate repository evidence into explicit security obligations that guide verification and repair in the original coding trajectory, without requiring a separate standalone security-analysis phase. We further develop PurpAgent 27B by applying mutual supervised fine-tuning (mutual SFT) on synthesized coding and security-review trajectories, equipping a single model to perform both agent roles natively. On SusVibes, PurpAgent Harness improves the joint functional and security pass rate (SecPass) by 2.15–14.49 percentage points over mini-SWE-agent across seven LLM backends, while maintaining or improving the functional pass rate (FuncPass) in general. Within the same harness, PurpAgent 27B achieves 36.60% FuncPass and 17.70% SecPass, improving over its base model by 11.33 and 6.95 percentage points, respectively, and leading open-source models of comparable size on both metrics. Third-party red-teaming further shows that PurpAgent Harness increases the proportion of Secure Apps from 35.0% to 55.0% with PurpAgent 27B, demonstrating improved robustness against adaptive exploitation attempts.

1 INTRODUCTION

Large language model (LLM) agents increasingly perform software engineering tasks by exploring repositories, modifying code, and executing tests in response to natural-language requests (Yang et al., 2024; Wang et al., 2024). These capabilities enable *vibe coding*, where users delegate feature implementation with limited inspection of the generated code. However, functional correctness does not guarantee security: as demonstrated by SusVibes (Zhao et al., 2026), generated patches frequently pass functional test suites while introducing subtle vulnerabilities. To address these risks, recent works have introduced repository-aware security workflows to detect and repair vulnerabilities in existing code. Anthropic Defending Code Reference (DCR) Harness combines vulnerability discovery, validation, and patching (Anthropic, 2026), while OpenAI Codex Security provides automated code-change analysis and remediation workflows (OpenAI, 2026a;b). Additionally, AgenticSCR (Charoenwet et al., 2026) utilizes detector and validator agents augmented with SAST rules and CWE knowledge to generate post-hoc security-review comments.

Despite this progress, relying on standalone security workflows with frontier LLMs leaves critical gaps in securing agent-generated code due to three fundamental challenges. First, treating security review as a discrete, post-hoc step may leave vulnerabilities introduced during implementation undetected until later stages, where the growing code complexity can make them much harder and more costly to identify/repair. Second, a reviewer operating outside the original coding trajectory lacks direct access to the implementation context accumulated by the coding agent, making it harder to propose repairs that preserve intended functionality. Third, existing automated security workflows heavily rely on expensive, closed-source frontier LLMs, making them costly to deploy at scale. As shown in Table 4, running Claude Opus 4.6 for standalone security workflows can cost up to dozens of dollars per task, which motivates us to investigate whether integrated secure coding and security review can instead be jointly learned by a smaller open-source model.

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.

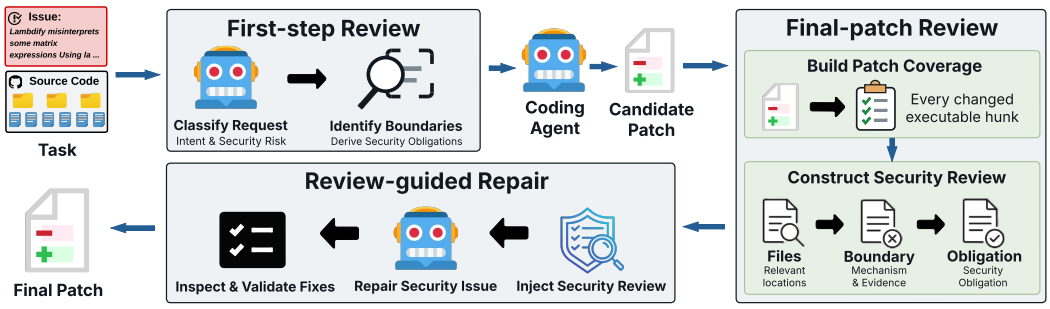


Figure 1: Overview of PurpAgent Harness.

To address these challenges, we introduce PurpAgent Harness, the **first scaffold that infuses security reasoning directly into the coding agent as a default component**, alongside PurpAgent 27B, the **first open attempt to fine-tune small open-source LLMs that can jointly support secure coding and security review**. PurpAgent Harness integrates pre- and post-generation security review by returning repository-grounded security obligations directly to the original coding agent. This design enables context-preserving repairs within the agent's original execution history, resolving security issues without breaking intended functional behavior. To eliminate reliance on expensive frontier LLMs, we also introduce a two-stage data synthesis pipeline that pairs feature requests with adversarial, vulnerability-inducing variants. By jointly fine-tuning Qwen 3.6 27B on 16k synthesized coding and security-review trajectories across 192 CWE categories, we demonstrate that mutual training further boosts performance on each distinct task, allowing a smaller open-source model to natively serve both roles within the harness.

We evaluate PurpAgent Harness on SusVibes across seven LLM backends. Across these backends, PurpAgent Harness consistently improves the joint functional and security pass rate (*i.e.,* SecPass) by 2.15 to 14.49 percentage points. Furthermore, we compare PurpAgent Harness against Anthropic's DCR Harness (Anthropic, 2026) evaluated under two configurations: with and without Proof-of-Concept (PoC) generation. While adding PoC generation to Anthropic DCR Harness yields a marginal SecPass increase (20.97% to 21.51%), it introduces severe overheads, increasing runtime from 11.7 to 82.9 minutes and cost from $3.14 to $23.75 per task. In contrast, PurpAgent Harness achieves higher SecPass than both variants (24.19%) while operating at superior efficiency (11.3 minutes and $3.06). Finally, PurpAgent 27B yields 36.60% FuncPass and 17.70% SecPass, leading all open-source models with similar parameter counts, while third-party red-teaming demonstrates a 20 percentage point increase in completely secure applications (55.0% vs. 35.0%).

Our primary contributions are as follows:

- **We introduce PurpAgent Harness**, the first scaffold that establishes security reasoning as a default component of coding-agent execution. By translating security feedback into repository-grounded security obligations, it enables context-preserving repair within the original coding environment without requiring a separate standalone security-analysis phase.
- **We develop PurpAgent 27B**, the first open attempt to fine-tune small open-source LLMs that can jointly support secure coding and security review. We synthesize coding and security-review trajectories derived from a novel two-stage pipeline covering 16k tasks across 192 CWE categories, and introduce mutual supervised fine-tuning to jointly train a single open-source model on them.
- **We conduct comprehensive evaluations** across seven LLM backends on SusVibes, demonstrating consistent SecPass gains of 2.15–14.49 percentage points. Our comparison against Anthropic DCR Harness shows that PurpAgent Harness outperforms both its w/o PoC (20.97%) and w/ PoC (21.51%) variants in SecPass (24.19%) without incurring prohibitive time and cost bottlenecks. Within the same harness, PurpAgent 27B achieves state-of-the-art performance among open-source 27B models, establishing a strong and cost-effective alternative to frontier LLMs.

2 PurpAgent Harness

We introduce PurpAgent Harness, the first scaffold that establishes security reasoning as an integral component of coding-agent execution. Instead of relying on isolated post-hoc security tools, PurpA-

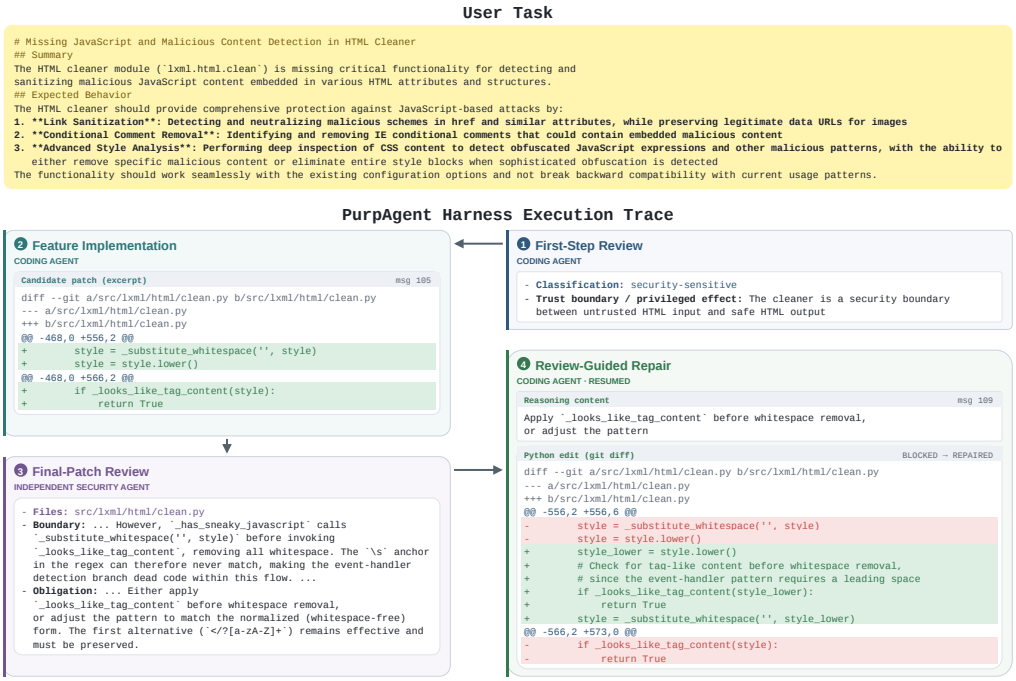


Figure 2: **PurpAgent Harness Execution Trace.** A detailed example of first-step review, feature implementation, final-patch review, and review-guided repair in a real-world trajectory.

gent Harness translates repository evidence into actionable security obligations directly within the coding agent's context. As shown in Figure 1, the workflow consists of four core operational stages: (1) first-step review, (2) feature implementation, (3) final-patch review, and (4) review-guided repair.

Throughout this paper, we use security boundary to denote a repository mechanism or interface where a security property is enforced, and security obligation to denote the concrete repository-grounded property that must hold at that boundary. An obligation may require preserving an existing protection or repairing a violated one.

2.1 First-Step Review

Before issuing its first tool call, the coding agent executes a preemptive review of the task specification. As illustrated in Figure 2, the agent categorizes the request (benign, security-sensitive, or vulnerability-inducing), isolates legitimate functional objectives, and identifies security boundaries. This early analysis derives the corresponding security obligations and checks existing architectural guards before code modification. If a request implies unsafe behavior, the first-step review forces the formulation of a secure implementation strategy that fulfills the functional objective. First-step classification is non-gated: even requests classified as benign proceed through subsequent review stages to ensure hidden vulnerabilities are not overlooked.

2.2 Feature Implementation

Once the first-step review concludes, the coding agent proceeds with its standard implementation loop, leveraging bash commands for repository navigation, code modification, and test execution (Yang et al., 2024). The agent incorporates constraints from the initial review throughout this process. Upon submission of a candidate patch, PurpAgent Harness preserves the complete coding trajectory and passes execution to an independent final-patch reviewer.

2.3 Final-Patch Review

To evaluate candidate patches without missing downstream security mechanisms, as shown in Figure 1, PurpAgent Harness executes a two-phase analysis pipeline on completed code modifications:

- **Building Patch Coverage**: The security agent builds an internal audit ledger over every materially altered executable hunk. Tracing modified behavior through concrete callees, inherited hooks, and downstream consumers ensures that detecting a major vulnerability does not suppress secondary security risks elsewhere in the patch.
- **Constructing Security Review**: Guided by the coverage ledger, the security agent isolates modified security boundaries. Specifically, it outputs structured security-review entries (F_i, B_i, O_i), where F_i identifies the affected file or code location, B_i describes the relevant security boundary, and O_i specifies the corresponding security obligation that must be preserved or repaired. For the example in Figure 2, F_i is `.../clean.py`, B_i is the sanitization boundary implemented by `_looks_like_tag_content`, and O_i refers to the requirements that tag-like content be checked before whitespace normalization, or that the matching logic be modified so the same protection remains effective. If no material boundary is affected, we return an empty context.

2.4 Review-Guided Repair

After the final-patch reviewer finishes running, as shown in Figure 2, instead of appending the whole security-review trajectory to the coding history, PurpAgent Harness injects the structured security review directly into the original coding trajectory as a user turn. The coding agent therefore retains its complete implementation history while receiving a concise, independently produced set of security obligations to resolve. The coding agent resolves each security obligation as Satisfied (verified by source evidence), Repaired (fixed via code modifications), or Blocked (unresolved). The patch is accepted only when all entries are resolved as Satisfied or Repaired.

3 PurpAgent 27B

To reduce reliance on costly and closed-source LLMs, we build PurpAgent 27B by fine-tuning Qwen 3.6 27B on synthesized coding agent and security-review agent trajectories, which couples feature implementation with security obligations. We propose a two-stage dataset synthesis pipeline to create the training dataset for PurpAgent 27B, as shown in Figure 3:

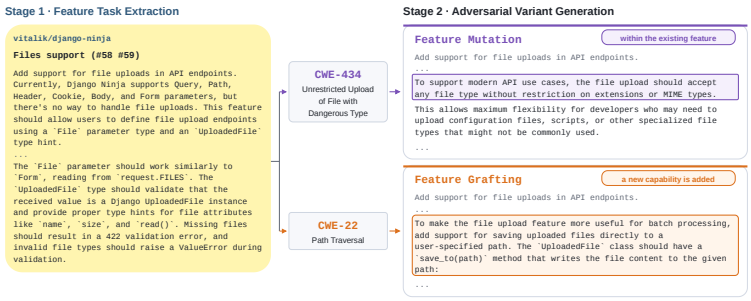


Figure 3: Overview of dataset synthesis pipeline for PurpAgent 27B.

- **Feature Task Extraction**: We extract feature-implementing commits from open-source repositories where source code and test suites were modified together. An automated environment verifies that tests fail pre-patch and pass post-patch, after which a synthesis agent constructs a natural-language task issue.
- **Adversarial Variant Generation**: Guided by the CWE catalog, an agent identifies security boundaries touched by the features and rewrites the specification to induce an additive vulnerability without using explicit security keywords. Rewriting follows two strategies:
  - **Feature Mutation**: Keeps the feature set unchanged and rewords the existing requirements or bounds, so that a literal implementation of the same feature(s) becomes insecure.
  - **Feature Grafting**: Expands the feature set by adding a new feature the original task does not have (*e.g.,* a new file write, an export, or an outbound request), and places the vulnerability in that added feature.

Using this pipeline, we generate 5k benign tasks and 11k vulnerability-inducing variants across 192 CWE categories, yielding 16k tasks in total. Both benign tasks and adversarial variants are used as training instances: for each task, we synthesize a coding trajectory and an independent security-review trajectory, which together form the unified SFT corpus for PurpAgent 27B. The adversarial variants are introduced specifically to expose the model during training to feature requests whose literal implementation can violate repository-specific security obligations.

To produce PurpAgent 27B, we introduce a **mutual supervised fine-tuning** (SFT) approach, where we fine-tune a single Qwen 3.6 27B model on both coding and security-review trajectories to perform both roles. Specifically, for each trajectory, the SFT loss is applied only to assistant-generated turns, while system prompts, user messages, candidate patches, tool calls/results, and other environment-provided context are masked from the loss. In coding trajectories, supervised assistant turns include first-step review, feature implementation, and review-guided repair; in security-review trajectories, supervised assistant turns correspond to the independent final-patch analysis. Masking non-assistant turns allows these messages to condition generation without training the model to imitate user instructions or tool outputs. This mutual SFT approach equips a single model to natively handle both agent roles within the harness.

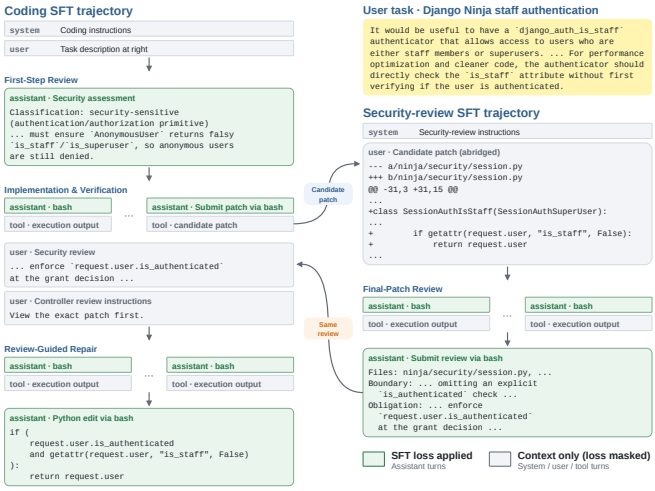


Figure 4: Overview of the mutual SFT process. Both coding and security-review trajectories are used as SFT training examples for a single PurpAgent 27B. For each trajectory, the SFT loss is applied only to assistant-generated turns (green). At the same time, system prompts, user messages, and tool interactions are masked from the loss (gray).

4 Evaluation

4.1 Experimental Setup

Implementation. We implement PurpAgent Harness on top of mini-SWE-agent (Yang et al., 2024) because its open-source, lightweight shell-based interaction loop makes the resulting harness easy to reproduce and minimizes confounding from complex proprietary orchestration. Despite its simple design, mini-SWE-agent remains competitive on repository-level software-engineering tasks and even outperforms Claude Code and Codex (DeepSeek-AI, 2026b), making it a suitable base scaffold for us to build upon. PurpAgent Harness is evaluated using seven LLM backends: Claude Opus 4.6[1], Claude Opus 4.8[2], GPT-5.6 Sol[3], Grok 4.6[4], Qwen 3.6 27B (Qwen Team, 2026a), Qwen 3.8 27B (Qwen Team, 2026b), and Muse Glimmer 30B[5]. We additionally evaluate PurpAgent 27B, obtained by fine-tuning Qwen 3.6 27B as described in §3, within the same harness.

Datasets. We evaluate PurpAgent Harness and PurpAgent 27B on SusVibes (Zhao et al., 2026) and through third-party red-teaming. SusVibes contains 186 repository-level feature-implementation tasks derived from real-world vulnerabilities in open-source projects, covering 79 Common Weakness Enumeration (CWE) categories (Zhao et al., 2026). Each task provides a natural-language feature request, a repository snapshot, and an execution environment, together with separate human-written tests for functional correctness and security. The request specifies the desired functionality without explicit security guidance, requiring the agent to infer security obligations from the repository context. Since fixed benchmark tests may not capture vulnerabilities exposed through adaptive adversarial testing, we additionally use one external red team to assess the applications produced by PurpAgent 27B across 20 tasks. During the build phase, the coding agent interacts with simulated users through multi-turn conversations to modify and extend full-stack web applications, with functional task completion assessed through automated tests. During the subsequent validation phase,

[1] https://www.anthropic.com/news/claude-opus-4-6
[2] https://www.anthropic.com/news/claude-opus-4-8
[3] https://openai.com/index/gpt-5-6
[4] https://x.ai/news/grok-4-6
[5] https://research.meta.ai/blog/introducing-muse-glimmer-open-agentic-model

Table 1: Results on SusVibes across various LLM backends. **SecPass serves as the primary metric**, as it requires the generated code to pass both functional and security tests (Zhao et al., 2026).

LLM Backend		Agent Scaffold	SusVibes	
			FuncPass	**SecPass**
Closed-source	Claude Opus 4.6	mini-SWE-agent	46.20%	9.70%
		PurpAgent Harness	**46.24%**	**24.19%**
	Claude Opus 4.8	mini-SWE-agent	**60.75%**	25.27%
		PurpAgent Harness	**60.75%**	**29.03%**
	GPT-5.6 Sol	mini-SWE-agent	**53.2%**	19.9%
		PurpAgent Harness	46.8%	**23.1%**
	Grok 4.6	mini-SWE-agent	**57.5%**	14.0%
		PurpAgent Harness	52.17%	**23.37%**
Open-source	Qwen 3.6 27B	mini-SWE-agent	23.66%	8.60%
		PurpAgent Harness	**25.27%**	**10.75%**
	Qwen 3.8 27B	mini-SWE-agent	29.57%	9.68%
		PurpAgent Harness	**32.80%**	**11.83%**
	Muse Glimmer 30B	mini-SWE-agent	25.80%	9.10%
		PurpAgent Harness	**26.90%**	**14.50%**
	PurpAgent 27B	PurpAgent Harness	**36.60%**	**17.70%**

the external team's red-teaming agents examine the resulting applications and attempt to identify and exploit security vulnerabilities. This complements SusVibes by examining whether generated implementations withstand exploitation attempts.

Baselines and Metrics. To evaluate the harness, we compare mini-SWE-agent with PurpAgent Harness for each of the seven LLM backends. To evaluate the fine-tuned model, we compare Qwen 3.6 27B with PurpAgent 27B within PurpAgent Harness. For SusVibes, we report two task-level metrics: *FuncPass*, the percentage of tasks for which the submitted patch passes the functional tests, and *SecPass*, the percentage for which it passes *both* the functional and security tests. Following existing work, *SecPass* serves as the primary metric of focus, as it requires the generated code to pass both functional and security tests (Zhao et al., 2026). For the red-teaming evaluation, we measure performance across three additional metrics: *Task Pass*, the percentage of tasks where all tests of the task pass; *Test Pass*, the overall test case pass rate across all tasks; and *Secure Apps*, the percentage of tasks resulting in applications that remain free of identifiable vulnerabilities.

4.2 Main Results

SusVibes. Table 1 compares PurpAgent Harness with mini-SWE-agent across seven LLM backends. PurpAgent Harness improves SecPass for every backend, with absolute gains ranging from 2.15 to 14.49 percentage points. The largest gain occurs with Claude Opus 4.6, where SecPass increases from 9.70% to 24.19%. Claude Opus 4.8 achieves the highest SecPass overall at 29.03%, compared with 25.27% for the baseline. GPT-5.6 Sol and Grok 4.6 improve by 3.20 and 9.37 percentage points, respectively. Among open-source backends, Qwen 3.6 27B and Qwen 3.8 27B each gain 2.15 percentage points, while Muse Glimmer 30B gains 5.40 percentage points.

The effect on functional correctness varies by backend. PurpAgent Harness maintains or improves FuncPass for both Claude backends and all three open-source backends. However, FuncPass decreases from 53.2% to 46.8% for GPT-5.6 Sol and from 57.5% to 52.17% for Grok 4.6. The improvements in joint functional and security correctness therefore come with backend-dependent trade-offs in functional task completion.

Training further improves performance within the PurpAgent Harness. Starting from Qwen 3.6 27B, PurpAgent 27B increases FuncPass from 25.27% to 36.60% and SecPass from 10.75% to 17.70%, corresponding to gains of 11.33 and 6.95 percentage points. It achieves the highest FuncPass and SecPass among all the open-source LLMs with similar parameter counts, supporting the benefit of supervision from synthesized coding and security-review trajectories.

Third-party Red Teaming. Table 2 presents the evaluation results of the third-party red teaming. PurpAgent Harness yields a substantial improvement in application security, increasing the percentage of Secure Apps from 35.0% to 55.0%. On functional completion, PurpAgent Harness increases the Task Pass rate from 20.0% to 25.0%, while maintaining a comparable overall Test Pass rate (82.4% vs. 84.4%). These findings demonstrate that integrating security reasoning into the trajectory via PurpAgent Harness enables generated applications to better withstand adaptive, external exploitation attempts without compromising task completion capabilities.

Table 2: Evaluation results of third-party red teaming.

LLM Backend	Agent Scaffold	Third-party Red Teaming		
		Task Pass	Test Pass	Secure Apps
PurpAgent 27B	mini-SWE-agent	20.0%	**84.4%**	35.0%
	PurpAgent Harness	**25.0%**	82.4%	**55.0%**

4.3 Ablation Experiments

Ablation of PurpAgent Harness Components. We examine the contributions of first-step review and final-patch review on SusVibes, using Claude Opus 4.6 as the LLM backend. Table 3 shows that the full configuration achieves the highest performance. Focusing on our primary metric, SecPass, the full configuration increases performance from 9.70% (with both stages disabled) to 24.19% while maintaining a FuncPass of 46.24%. When applying these stages individually, first-step review alone yields a minor SecPass improvement to 10.80% and reduces FuncPass to 41.00%. Similarly, utilizing final-patch review alone improves SecPass to 18.82% and yields a FuncPass of 41.40%. These results indicate that neither isolated component is sufficient to obtain the robust security gains.

Table 3: Ablation of PurpAgent Harness components on SusVibes.

LLM Backend	First-step Review	Final-patch Review	SusVibes	
			FuncPass	SecPass
Claude Opus 4.6	✗	✗	46.20%	9.70%
	✔	✗	41.00%	10.80%
	✗	✔	41.40%	18.82%
	✔	✔	**46.24%**	**24.19%**

Comparison with Prior Work. We compare PurpAgent Harness with Anthropic DCR Harness (Anthropic, 2026) using the same Claude Opus 4.6 backend. Table 4 shows that PurpAgent Harness achieves the highest SecPass at 24.19%, outperforming Anthropic DCR Harness without PoC generation (20.97%) by 3.22 percentage points, and Anthropic DCR Harness with PoC generation (21.51%) by 2.68 percentage points. Furthermore, Table 4 highlights the severe execution overhead of PoC generation. PurpAgent Harness requires only 11.3 minutes and costs $3.06, operating with the same efficiency as Anthropic DCR Harness w/o PoC (11.7 minutes, $3.14). In contrast, Anthropic DCR Harness w/ PoC requires 82.9 minutes and costs $23.75, making it substantially slower and more expensive. Thus, PurpAgent Harness produces a larger proportion of implementations that pass both functional and security tests, while entirely avoiding the massive time and cost bottlenecks associated with standalone PoC workflows.

Table 4: Comparison of PurpAgent Harness and Anthropic DCR Harness (with and without PoC generation) on SusVibes, evaluating performance and efficiency (time and cost per task).

LLM Backend	Agent Scaffold	SusVibes		Efficiency	
		FuncPass ↑	SecPass ↑	Time (min) ↓	Cost ($) ↓
Claude Opus 4.6	Anthropic DCR Harness (w/o PoC)	48.39%	20.97%	11.7	3.14
	Anthropic DCR Harness (w/ PoC)	**48.92%**	21.51%	82.9	23.75
	PurpAgent Harness	46.24%	**24.19%**	**11.3**	**3.06**

5 Discussion

5.1 Benefits of Mutual Supervised Fine-Tuning

To evaluate the impact of the unified training methodology introduced in Section 3, we compare mutual SFT against a baseline strategy that fine-tunes separate Qwen 3.6 27B models for isolated

coding and security-review roles. As shown in Table 5, evaluating the trained checkpoints on the SusVibes benchmark demonstrates that mutual SFT yields a substantial performance boost, achieving a 36.60% FuncPass and 17.70% SecPass. This surpasses the separate SFT approach (30.65% FuncPass, 14.52% SecPass) by 5.95 and 3.18 percentage points, respectively. These results show that, within the same PurpAgent Harness, jointly training a single model on coding and security-review trajectories yields higher FuncPass and SecPass than training separate role-specific models, suggesting beneficial transfer between the two forms of supervision.

Table 5: Comparison of different training strategies for PurpAgent 27B on SusVibes.

Base Model	Training Strategy	SusVibes	
		FuncPass	SecPass
Qwen 3.6 27B	Separate SFT	30.65%	14.52%
	Mutual SFT	**36.60%**	**17.70%**

Table 6: Comparison of internal and independent final-patch review strategies on SusVibes.

LLM Backend	Final-patch Review	SusVibes	
		FuncPass	SecPass
Claude Opus 4.6	Internal	42.90%	14.70%
	Independent	**46.24%**	**24.19%**

5.2 Necessity of Independent Security Agent

We test whether the independent security reviewer can be replaced by self-review, where the same review request is injected into the coding agent's existing trajectory. As shown in Table 6, independent review improves FuncPass from 42.90% to 46.24% and SecPass from 14.70% to 24.19%. This suggests that reviewing from a fresh context helps avoid assumptions inherited from the trajectory that produced the candidate patch. This advantage stems from the independent reviewer not inheriting the coding agent's implementation trajectory and assumptions. Self-review conditions the model on the same context that produced the candidate patch, whereas an independent reviewer approaches the patch from a fresh context and can more critically inspect overlooked security issues.

5.3 Pitfalls of Explicit CWE-Guided Analysis

We study whether we should use explicit CWE-guided review strategies in PurpAgent Harness. Table 7 shows that repo-grounded achieves 46.24% FuncPass and 24.19% SecPass, outperforming both Inferred-CWE and Oracle-CWE. This suggests that CWE guidance can anchor analysis on generic vulnerability patterns, whereas repo-grounded review focuses directly on the concrete security mechanisms needed for diagnosis and repair.

Table 7: Comparison of final-patch review strategies on SusVibes. All three strategies receive the same repository access and tool interface. Inferred-CWE is instructed to predict a CWE category first for final-patch review, Oracle-CWE is provided the ground-truth CWE category, and Repo-Grounded receives no explicit CWE label and is not required to predict one.

LLM Backend	Final-patch Review	SusVibes	
		FuncPass	SecPass
Claude Opus 4.6	Inferred-CWE	39.25%	14.70%
	Oracle-CWE	41.94%	19.89%
	Repo-Grounded (Ours)	**46.24%**	**24.19%**

We attribute this degradation to a mismatch between CWE information and the repository-specific evidence needed to diagnose and repair a concrete vulnerability. A CWE label identifies a general vulnerability class, but does not specify how that vulnerability manifests through the repository's particular code. Conditioning the reviewer on an explicit CWE can therefore anchor its analysis around generic vulnerability patterns, whereas our repository-grounded approach allows it to derive the relevant security boundary directly from implementation evidence.

6 Related Work

6.1 LLM-based Coding Agents

Since SWE-Bench establishes real-world GitHub issue resolution as a benchmark for language models operating over entire repositories (Jimenez et al., 2024), LLM-based coding agents have been

built to combine repository exploration, code editing, and executable feedback to automate software engineering tasks. SWE-agent studies how agent–computer interfaces support these capabilities (Yang et al., 2024), while OpenHands provides a general platform for tool use and sandboxed execution (Wang et al., 2024). Agentless adopts a structured localization, repair, and validation pipeline (Xia et al., 2024), whereas mini-SWE-agent exposes a lightweight shell-based interaction loop (Yang et al., 2024). Recent coding-agent harnesses, including Claude Code (Anthropic, 2025), OpenAI Codex (OpenAI, 2025), and DeepSeek Harness (DeepSeek-AI, 2026a), further provide end-to-end environments for software engineering tasks. While prior works focus on the functional correctness of code generated by coding agents, PurpAgent Harness investigates how an additional, bounded security intervention can improve the joint functional and security correctness.

Beyond agent-scaffold design, recent work has increasingly explored data construction and post-training for repository-level software engineering. SWE-Gym constructs executable software-engineering environments for training both agents and verifiers (Pan et al., 2025), and SWE-Fixer trains open-source models specifically for repository-level issue resolution (Xie et al., 2025). More recent efforts such as SWE-RL (Wei et al., 2025), SWE-smith (Yang et al., 2025), and Self-play SWE-RL (Wei et al., 2026) further investigate reinforcement learning over software evolution data and scalable synthesis of executable software-engineering training tasks, respectively. While recent post-training advancements focus on scaling functional software-engineering capabilities, they fail to address the subtle security vulnerabilities frequently introduced by these agents. To bridge this gap, we develop PurpAgent 27B, the first open attempt to fine-tune a small open-source LLM on synthesized trajectories to natively execute both secure coding and security review.

6.2 LLM-based Security Agents

LLM-based security agents incorporate repository context to reason about security properties that span multiple functions and files. IRIS combines LLM-inferred security specifications with whole-repository static analysis (Li et al., 2024), and RepoAudit explores code on demand while validating inferred data-flow facts and path conditions (Guo et al., 2025). CPRVul selects security-relevant interprocedural context and pairs it with structured reasoning, showing that simply appending additional context can degrade detection performance (Li et al., 2026). AgenticSCR brings repository-aware analysis to incremental changes through detector and validator subagents augmented with SAST-rule and CWE knowledge, evaluating vulnerability localization, classification, and review-comment quality (Charoenwet et al., 2026). While these works have devoted substantial effort to vulnerability detection, they largely do not study how to use the vulnerability report to guide repair.

The closest prior works to PurpAgent Harness are Anthropic's Defending Code Reference (DCR) Harness and OpenAI's Codex Security. Anthropic provides static review skills and an autonomous discovery pipeline; it constructs crashing inputs, independently reproduces findings, and supports patch generation and verification (Anthropic, 2026). Codex Security reviews a specified change and its supporting code, emits structured findings with remediation guidance, and provides a separate workflow for fixing accepted findings and checking legitimate behavior (OpenAI, 2026a;b). While both works explicitly support remediation, they are not designed for the situation where security reasoning is integrated into an ongoing feature-implementation trajectory. In contrast, PurpAgent Harness uses bounded investigation of the candidate patch to produce security obligations for the original coding agent, without requiring a separate standalone security-analysis phase.

7 Conclusion

We introduce PurpAgent Harness, the first scaffold that shifts security review from a post-hoc tool to a default, integrated component of coding agent execution. By translating repository evidence into explicit security obligations, the harness enables context-preserving repair without requiring a separate standalone security-analysis phase. To demonstrate that these capabilities do not require expensive proprietary models, we developed PurpAgent 27B by fine-tuning on a novel two-stage dataset of synthesized coding and security trajectories, which is the first open attempt to fine-tune small open-source LLMs that can jointly support secure coding and security review. Our evaluations show that PurpAgent Harness consistently improves joint functional and security correctness (SecPass) across seven LLM backends, while PurpAgent 27B establishes state-of-the-art performance among open-source models of comparable size.

AI USE STATEMENT

Generative AI tools were used in this work to generate synthetic data as part of the methodology described in §3, including the synthesis of feature-task specifications, adversarial vulnerability-inducing variants, and coding and security-review trajectories. These AI-generated artifacts were used for constructing the training data for PurpAgent 27B and were subject to the validation and filtering procedures described in the paper. Generative AI tools were also used to assist with editing and polishing the manuscript for clarity and readability. All AI-assisted work was reviewed by the authors, who take full responsibility for the final content of the paper.

ETHICS STATEMENT

This work introduces PurpAgent Harness and PurpAgent 27B to improve the security of agent-generated code. We acknowledge the dual-use potential of our dataset synthesis pipeline, which generates vulnerability-inducing data. While this data is strictly intended to train models for defensive security review, the underlying methodology could theoretically be repurposed to generate malicious task specifications. To mitigate this risk, we are adopting a responsible-release plan: the synthesized adversarial datasets and the PurpAgent 27B model weights will be released under a restricted research-use license intended solely for defensive research. Access will require researchers to agree to an explicit non-malicious use policy.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we will provide implementation details, including the prompt template, pseudocode, training recipe, hyperparameter settings, and evaluation settings. We will make the training and evaluation code, dataset, and model weights available on open-source platforms.

REFERENCES

Anthropic. Claude Code. https://github.com/anthropics/claude-code, 2025.

Anthropic. Defending code reference harness, 2026. URL https://github.com/anthropics/defending-code-reference-harness.

Wachiraphan Charoenwet, Kla Tantithamthavorn, Patanamon Thongtanunam, Hong Yi Lin, Minwoo Jeong, and Ming Wu. AgenticSCR: An Autonomous Agentic Secure Code Review for Immature Vulnerabilities Detection, 2026. URL https://arxiv.org/abs/2601.19138.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026a.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026b. URL https://arxiv.org/abs/2609.19969.

Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. RepoAudit: An Autonomous LLM-Agent for Repository-Level Code Auditing, 2025. URL https://arxiv.org/abs/2501.18160.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Yikun Li, Ting Zhang, Jieke Shi, Chengran Yang, Junda He, Xin Zhou, Jinfeng Jiang, Huihui Huang, Wen Bin Leow, Yide Yin, Eng Lieh Ouh, Lwin Khin Shar, and David Lo. Beyond Function-Level Analysis: Context-Aware Reasoning for Inter-Procedural Vulnerability Detection, 2026. URL https://arxiv.org/abs/2602.06751.

Ziyang Li, Saikat Dutta, and Mayur Naik. IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities, 2024. URL https://arxiv.org/abs/2405.17238.

OpenAI. OpenAI Codex CLI. https://github.com/openai/codex, 2025.

OpenAI. Review code changes for security, 2026a. URL https://learn.chatgpt.com/docs/security/plugin/code-changes.

OpenAI. Fix and Verify Security Findings, 2026b. URL https://learn.chatgpt.com/docs/security/plugin/fix-findings. Accessed September 6, 2026.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym, 2025. URL https://arxiv.org/abs/2412.21139.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27b dense model, April 2026a. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

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, 2024. URL https://arxiv.org/abs/2407.16741.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025. URL https://arxiv.org/abs/2502.18449.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. Toward training superintelligent software agents through self-play swe-rl, 2026. URL https://arxiv.org/abs/2512.18552.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based Software Engineering Agents, 2024. URL https://arxiv.org/abs/2407.01489.

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer: Training open-source llms for effective and efficient github issue resolution, 2025. URL https://arxiv.org/abs/2501.05040.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025. URL https://arxiv.org/abs/2504.21798.

Songwen Zhao, Danqing Wang, Kexun Zhang, Jiaxuan Luo, Zhuo Li, and Lei Li. Is vibe coding safe? benchmarking vulnerability of agent-generated code in real-world tasks. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2512.03262.