When AI agents perform silent refactoring without cryptographic provenance checks, does the process actively overwrite architectural intent tracking by severing original design constraints and rewriting implicit contracts, or does it preserve the initial intent structure while accumulating unverified drift that only manifests during human review?

July 17, 2026 | SnugLab Research

Executive Summary

When AI agents perform silent refactoring without cryptographic provenance checks, the process actively overwrites architectural intent by severing original design constraints and rewriting implicit contracts, rather than merely preserving the initial intent structure while accumulating unverified drift. Evidence suggests that AI agents, lacking explicit architectural guidance and cryptographic provenance, treat explicit constraints and implicit patterns as equivalent statistical signals, leading to immediate architectural violations that subsequently manifest as accumulated, unverified drift during human review. The key unresolved uncertainty lies in precisely quantifying the direct impact of "active overwrites" versus the "accumulation of drift" across diverse architectural patterns, as the two phenomena are deeply intertwined and often co-occur. Specific empirical studies with controlled architectural constraints would further clarify this distinction.

Key Findings

AI Agents Actively Overwrite Architectural Intent, Leading to Drift

AI agents, when performing silent refactoring without cryptographic provenance, actively overwrite architectural intent by severing original design constraints and rewriting implicit contracts [1, 7, 8]. This occurs because agents optimize for immediate task completion, test satisfaction, and effort minimization, rather than long-term architectural coherence [7, 8]. They disregard established abstractions, inject concrete infrastructure types into application code, and duplicate business rules across different layers [7]. For example, AI agents frequently alter core business logic to simplify code, directly overwriting the program's fundamental rules [1, 6]. This active alteration causes "semantic lineage

erosion," breaking the causal links between code changes, architectural intent, and dependency graphs [3].

This active overwriting subsequently manifests as unverified drift that accumulates over time [3, 4]. AI agents predominantly perform localized, low-level changes, such as renaming variables (8.5%), renaming parameters (10.4%), and changing variable types (11.8%) [1]. While these modifications may initially preserve functional behavior and pass basic tests, they introduce subtle, unverified semantic deviations that diverge from the original design [4]. This hidden drift accumulates as technical debt and persistent vulnerabilities, only becoming apparent during human review or under real-world conditions when the unverified semantic changes cause failures [3, 4]. The historical track record of early AI coding deployments (2024-2025) demonstrates that silent refactoring involves both active architectural overwrites and the accumulation of unverified drift, which often remains undetected without cryptographic provenance [3, 4, 7, 8].

Cryptographic Provenance Primarily Detects Drift, While Explicit Constraints Prevent Overwrites

Implementing cryptographic provenance checks mainly serves as a detection mechanism that surfaces accumulated drift earlier in the development lifecycle, rather than actively preventing architectural overwrites through deterministic validation [2, 3, 9, 10]. Cryptographic provenance frameworks establish authenticity by hashing generated content and signing it with metadata like model identifiers and timestamps [2]. However, this only establishes the "who" and "when" of changes, inherently failing to capture the "why" behind the semantic reasoning and architectural intent [3]. As the Content Authenticity Initiative notes, "Relying solely on cryptographic identity binding inherently fails to capture the semantic reasoning and architectural intent behind patches" [3].

To actively prevent overwrites and enforce deterministic validation, systems must use explicit architectural constraints and architecture tests that enforce structural patterns at build time [7, 8]. When paired with metadata tracking tools, cryptographic provenance shifts quality and security checks earlier in the development lifecycle, moving validation into the "inner loop" at the moment code is generated [10]. This early detection mechanism surfaces hidden drift, accountability gaps, and security vulnerabilities before they compound into structural regressions during human review [3, 9]. Comprehensive provenance frameworks ultimately require tracking both cryptographic identity and semantic reasoning to fully govern AI-generated code [3].

Architectural Patterns Most Susceptible to Overwrites and Quantified Error Rates

Layered architecture and core-project patterns are most susceptible to silent refactoring overwrites [7]. AI agents frequently violate layered architecture by injecting concrete infrastructure types directly into application code, such as a `SendGridClient` [7]. They also disrupt core-project patterns by duplicating business rules and scattering logic across different layers instead of containing it within a single core project [7].

While specific error rates for individual architectural patterns are not quantified, general error rates for AI refactoring in 2024-2025 deployments are significant. AI agents fail approximately 70% of the time when performing multi-step tasks [3]. In 2025, AI tools assisted in generating 41% of new code merged globally [9]. However, 45% of AI-generated code samples fail security tests by introducing OWASP Top 10 vulnerabilities, and AI-generated code contains 2.5 times more high-severity vulnerabilities than human-written code [9].

Empirical Evidence Distinguishing Overwrites from Drift

Empirical evidence from various projects distinguishes immediate architectural constraint overwrites from preserved structures with cumulative semantic drift. In the `SysMARA` open-source TypeScript project, code generation without explicit, machine-traversable architectural constraints resulted in a 50% architectural violation rate, which dropped to 0% when such constraints were applied [14]. This demonstrates that explicit constraints can prevent immediate architectural overwrites.

Analysis of Hugging Face model repositories revealed instances of "optimized drift," where accuracy improved over commits, and "semantic preservation," where accuracy remained stable [11]. Conversely, some models experienced performance degradation after refactoring, indicating semantic drift away from intended performance [11]. The `Rel(AI)Build` project found that agent configurations propagate as undeclared shared components, with 10.1% exact duplicates across independent repositories, highlighting how unmanaged configurations lead to cumulative semantic drift and architectural sprawl [13]. Benchmarks like `EvoClaw` show that AI agents struggle to limit error accumulation and unmanaged architectural drift in continuous settings, with performance dropping from over 80% on isolated tasks to at most 38% in continuous settings [12].

AI Agent Refactoring Strategies and Context Limitations

Leading AI coding agents, including GitHub Copilot, Cursor, and Claude Code, default to refactoring strategies focused on low-level, consistency-oriented edits rather than high-level structural changes [1]. Their modifications predominantly include changing variable types (11.8%), renaming parameters (10.4%), and renaming variables (8.5%) [1]. These agents perform fewer high-level structural changes than human developers, accounting for 43.0% of agent changes compared to 54.9% for humans [1].

Unconstrained AI agents actively overwrite implicit contracts by disregarding established abstractions and injecting concrete infrastructure into inappropriate layers [7]. They often "try to be too smart," overriding business logic and simplifying code to the point of altering core program rules [1, 6]. This active overwriting causes "semantic lineage erosion," breaking the causal links between code changes, architectural intent, and dependency graphs [3]. While some tools like Cursor's Composer mode and JetBrains' Junie integrate with IDE static analysis to maintain codebase-wide context and enforce conventions, many agents still struggle with broader architectural goals and contextual blindness [2, 8]. For instance, Claude models typically allow for a 200K token context window, and GitHub Copilot's context window was increased to 192K tokens, but effective context can be smaller, and performance degrades noticeably past approximately 100K tokens [2, 5].

Implications

The findings indicate that silent AI refactoring without cryptographic provenance poses a significant risk to software architecture. The active overwriting of architectural intent means that the foundational design principles of a system can be fundamentally altered without explicit human intervention or awareness. This leads to an accumulation of unverified drift, which manifests as technical debt, increased complexity, and security vulnerabilities that are difficult to detect until late in the development cycle or during real-world operation [3, 4, 8, 9]. For development teams, this implies a critical need for robust architectural guardrails, explicit machine-readable constraints, and comprehensive provenance frameworks that track not only the "who" and "when" of changes but also the "why" (semantic reasoning) [3, 7, 8]. Relying solely on human review to catch these issues is insufficient, as the drift can be subtle and pervasive, leading to costly remediation efforts [3, 4].

Limitations and Caveats

The confidence in distinguishing between "active overwrites" and "accumulated drift" is moderate, as the two phenomena are often intertwined, with active overwrites directly leading to drift. Methodological debate exists on how to precisely define and measure these concepts empirically. While the evidence strongly suggests active overwriting is the primary driver, the exact quantification of its immediate impact versus the gradual accumulation of drift across diverse architectural patterns remains an area for further research. Specific error rates for individual architectural patterns are not consistently quantified across the available research. Furthermore, the computational overhead percentages for implementing cryptographic provenance mechanisms in CI/CD pipelines are not provided in the reviewed findings. Some numerical claims, particularly those from preprint or lower-tier sources, would benefit from corroboration by peer-reviewed studies or official industry reports.

Sources

[1] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2511.04824v1

[2] Download - ijaidsml.org - http://ijaidsml.org/index.php/ijaidsml/article/download/463/425

[3] Ais Hidden Drift Costs Millions - readme.snuglab.com - https://readme.snuglab.com/ais-hidden-drift-costs-millions/

[4] IDE Jonathan 2025 - seal-queensu.github.io - https://seal-queensu.github.io/publications/pdf/IDE-Jonathan-2025.pdf

[5] Ai Driven Code Refactoring For Legacy Apps - devoxsoftware.com - https://devoxsoftware.com/legacy-modernization/ai-solution-accelerator/ai-driven-code-refactoring-for-legacy-apps/

[6] [blog] Refactoring With Ai Agent B715ed32296d - vasili-manko.medium.com - https://vasili-manko.medium.com/refactoring-with-ai-agent-b715ed32296d

[7] Ai Agents Clean Architecture - blog.nimblepros.com - https://blog.nimblepros.com/blogs/ai-agents-clean-architecture/

[8] Architectural Constraints For Ai Agents - bitloops.com - https://bitloops.com/resources/governance/architectural-constraints-for-ai-agents

[9] Ai Code Provenance The Accountability Gap Nobody Has Closed - thinkscientist.com - https://www.thinkscientist.com/ai-code-provenance-the-accountability-gap-nobody-has-closed/

[10] Ai Coding Agents Are Pulling Ci Feedback Into The Inner Loop - devops.com - https://devops.com/ai-coding-agents-are-pulling-ci-feedback-into-the-inner-loop/

[11] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2512.07983v1

[12] Papers - huggingface.co - https://huggingface.co/papers?q=query%20capability%20deconstruction

[13] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2606.26924v1

[14] Article - americanimpactreview.com - https://americanimpactreview.com/article/e2026043