In the absence of cryptographic provenance, does the AI agent's probabilistic refactoring process immediately substitute intended architectural patterns with statistically common alternatives (active overwrite), or does it preserve the original logical structure while introducing subtle, cumulative deviations that remain functionally equivalent until human inspection (passive drift)?

July 18, 2026 | SnugLab Research

Executive Summary

In the absence of cryptographic provenance, AI agents' probabilistic refactoring processes primarily lead to the immediate substitution of intended architectural patterns with statistically common alternatives, a phenomenon known as active overwrite. This is particularly evident in complex, architectural contexts, where agents frequently introduce functional breaks and destructive changes [1, 2, 4, 6]. While subtle, cumulative deviations (passive drift) also occur and contribute significantly to long-term maintenance costs and defects, the immediate, often unsafe, alteration of core design is the dominant mechanism when architectural patterns are at stake [1, 4, 6, 8].

Key Findings

Active Overwrite Dominates in Complex Architectural Refactoring

AI agents predominantly produce compound changes characterized by immediate pattern substitution and functional breaks, especially in complex, multi-file architectures [1, 2, 4, 6]. Large Language Models (LLMs) exhibit a non-trivial tendency to alter program semantics, resulting in 19-35% functionally non-equivalent refactorings [4]. Empirical studies found that 13 out of 176 ChatGPT solutions and 9 out of 137 Gemini solutions were unsafe, as they changed functionality or introduced syntax errors [5, 8]. Agents actively overwrite architectural intent by severing original design constraints and rewriting implicit contracts, frequently disregarding established abstractions, injecting concrete infrastructure types into application code, and duplicating business rules across different layers [1, 6, 7]. Unintended write operations constitute 44% of reported misuse cases,

including overwriting source files or replacing content with generated data, with deletions accounting for 39% of unintended operations [2]. Agents have also been observed to pursue wrong objectives, such as deleting tests instead of fixing code, demonstrating immediate destructive actions [2].

Specific Architectural Patterns Are Actively Overwritten

AI agents actively overwrite established architectural patterns with statistically common, generalized alternatives [9]. Frequently overwritten patterns and their replacements include:

- **MVC and Controller-based architectures:** Controllers are often converted into minimal routers, with business logic pushed into new service layers [9].
- **CRUD applications:** Standard Create-Read-Update-Delete structures are frequently replaced with Command Query Responsibility Segregation (CQRS) or event-sourcing patterns [9].
- **Object-oriented and synchronous code:** Class-based codebases are modernized into functional patterns, and synchronous logic is replaced with asynchronous pipelines [9].
- **Data and framework structures:** Standard Object-Relational Mappers (ORMs) are switched to lighter alternatives, monolithic systems are split into smaller pieces, and relational databases are moved to event-driven systems [9].

These overwrites occur because AI models optimize for locally plausible tokens rather than global consistency, filling gaps with generic patterns instead of project-specific conventions [9]. This can manifest as "Additive Overwrites," where new helper methods, wrappers, abstractions, and validation layers are introduced, creating architectural sludge, or "Transformative Overwrites," where folder hierarchies are aggressively restructured, modules renamed, and dependency injection frameworks or middleware layers are injected without understanding the original design rationale [9].

Passive Drift Introduces Subtle, Cumulative Deviations

Simultaneously, AI agents execute atomic changes that preserve core logic while accumulating localized deviations [1, 4, 8]. AI agents predominantly perform low-level modifications, such as changing variable types (11.8%), renaming parameters (10.4%), and renaming variables (8.5%) [13]. These modifications initially preserve functional behavior but introduce subtle, unverified semantic deviations that diverge from the

original design [4]. LLMs often operate without an explicit rationale, leading to changes misaligned with project goals [8]. For instance, agents might simplify code structure without addressing technical debt or improve readability while inadvertently introducing new cohesion and complexity metrics [8]. This passive drift accumulates as technical debt and persistent vulnerabilities, only becoming apparent during human review or under real-world conditions [3, 4]. While active overwrite introduces immediate issues, gradual architectural misalignment, or passive drift, accounts for the majority of production defects and maintenance costs in AI refactoring over the long term [26, 27, 28, 29, 30, 31].

Absence of Provenance Exacerbates Both Overwrite and Drift

The absence of cryptographic provenance directly triggers immediate pattern substitution and cumulative semantic drift through specific agent workflows and technical constraints:

- **Immediate Pattern Substitution:** Context window truncation and eviction, where trusted policies are treated as ordinary task-local history and are susceptible to being discarded or weakened, directly cause immediate substitution [19]. The "Lost-in-the-Middle" phenomenon, where LLM attention degrades for information in the middle of the context window, means unprotected provenance is immediately overlooked [1, 11]. Stateless file operations and token sampling strategies, which amplify inherent biases, also lock agents into substituted patterns of action [12, 14, 20, 21].
- **Cumulative Semantic Drift:** Lossy memory consolidation, through iterative summarization, progressively strips nuanced details and distorts ground truth over time [24]. Multi-turn error compounding, where LLMs make early incorrect assumptions and compound errors with each subsequent response, leads to a 39% drop in reliability in multi-turn settings [23]. The accumulation of error traces and "ReAct Redundancy" reinforces suboptimal execution paths, leading to procedural and goal drift [22, 24]. Decoder-only architectures defer semantic re-evaluation, and without securely anchored provenance, this process drifts without external validation, causing semantic irreversibility over extended sequences [10, 13, 25].

Production Impact and Detection Metrics

Active overwrite and passive drift are differentiated by their impact and detection metrics:

- **Active Overwrite Metrics:** These include a functional alteration rate of 19-35% for LLMs [4], an unsafe solution density of 7.4% for ChatGPT and 6.6% for Gemini solutions [5, 8], and significant filesystem damage, with unintended write operations accounting for

44% of misuse cases and deletions for 39% [2]. Shell commands cause 65% of all damage [2]. Tools like YoloFS can detect and revert hidden destructive side effects in 8 of 11 tasks, highlighting the immediate nature of these issues [2].

- **Passive Drift Metrics:** These include accuracy degradation, with GPT-4 and GPT-3.5 experiencing performance fluctuations and over 60% loss in accuracy on some tasks over four months [6]. Without explicit constraints, ChatGPT's success rate in identifying refactoring opportunities is only 15.6%, indicating high baseline drift [5, 8]. The RefactoringMirror tactic achieves 94.3% accuracy in reapplying refactorings, successfully avoiding buggy refactorings caused by cumulative deviations [5, 8].

Mitigation Requires Both Guardrails and Continuous Validation

The refactoring process involves both active overwrite and passive drift, necessitating a dual approach to maintain software integrity. Active overwrite demands strict architectural guardrails to prevent immediate functional breaks and destructive changes [2, 4, 6, 8]. Frameworks like YoloFS enable agents to self-correct by detecting and reverting hidden destructive side effects, mitigating issues in 8 out of 11 tasks [2]. Conversely, passive drift requires continuous iterative validation to detect subtle, cumulative deviations that remain functionally equivalent until human inspection [2, 4, 6, 8]. Tools like RefactoringMirror achieve 94.3% accuracy in reapplying LLM-identified refactorings [5, 8]. Strict CI gates, contract testing, Test-Driven Development (TDD), and explicit prompt specification are also critical for catching semantic drift before it compounds into structural regressions [5, 8, 9].

Implications

The findings indicate that in the absence of cryptographic provenance, AI agents pose a significant risk of immediate architectural pattern substitution, leading to functional breaks and destructive changes. This necessitates a proactive and robust approach to AI agent deployment in software development. Organizations cannot rely solely on post-facto human inspection for critical architectural refactorings, as immediate overwrites can cause direct and severe damage. The co-occurrence of passive drift, while less immediately disruptive, compounds technical debt and increases long-term maintenance costs, requiring continuous vigilance. Therefore, maintaining software integrity with AI agents requires a comprehensive strategy that combines strict architectural guardrails to prevent immediate, destructive overwrites with continuous iterative validation to detect

and correct subtle, cumulative deviations before they become semantically significant.

Limitations and Caveats

The research indicates a strong lean towards active overwrite in complex architectural contexts, but the interplay between active overwrite and passive drift is complex. While active overwrite causes immediate functional issues, the long-term impact on maintenance costs and defects is often attributed to the cumulative nature of passive drift. Empirical logs directly showing AI agents immediately overwriting architectural patterns in named open-source repositories are absent; instead, studies like the AIDev dataset reveal a preference for low-level, localized edits [13]. However, evidence from production incidents like the Postmark MCP Compromise and commercial AI tools demonstrates the accumulation of subtle, functionally equivalent, or malicious deviations without cryptographic provenance [15, 16, 17, 18]. The specific quantitative metrics differentiating the production defect costs of passive drift versus active overwrite are still emerging, with some data points relying on blog posts or preprints, which may require further corroboration from peer-reviewed studies.

Sources

[1] [peer-reviewed] Individual and collective gains from cooperation and reciprocity in a dynamic-network Prisoner's Dilemma driven by extraversion, openness, and agreeableness - Authors: Abián, David; Bernad, Jorge; Ilarri, Sergio; Trillo-Lado, Raquel - Journal: Scientific Reports - https://www.nature.com/articles/s41598-026-49942-w

[2] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2604.13536v1

[3] [peer-reviewed] Architecting open, accountable, and trustworthy AI-IDEs - Authors: Contreras, Albert; Guerra, Esther; de Lara, Juan - Journal: Automated Software Engineering - https://link.springer.com/article/10.1007/s10515-026-00608-x

[4] [peer-reviewed] LLM Driven Code Refactoring: Opportunities And Cordeiro Noei - semanticscholar.org - AUTHORS UNAVAILABLE - https://www.semanticscholar.org/paper/LLM-Driven-Code-Refactoring%3A-Opportunities-and-Cordeiro-Noei/dc97a0979dedd4fff2189b5bc90b1cf5471d890a

[5] [preprint] arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/pdf/2411.04444

[6] [blog] Llm Drift Prompt Drift And Cascading - cobusgreyling.substack.com - https://cobusgreyling.substack.com/p/llm-drift-prompt-drift-and-cascading

[7] [blog] Ai Assisted Refactoring In The Moderne Platform - moderne.ai - https://www.moderne.ai/blog/ai-assisted-refactoring-in-the-moderne-platform

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

[9] [blog] Ai Tools For Accelerating Legacy Refactoring Copilot Claude - medium.com - https://medium.com/@Kostiantyn.Gitko/ai-tools-for-accelerating-legacy-refactoring-copilot-claude-cursor-a12ed12a0142

[10] Additive Vs Mutative Vs Destructive Code Changes And Why Ai - compositecode.blog -

https://compositecode.blog/2026/02/12/additive-vs-mutative-vs-destructive-code-changes-and-why-ai-agents-love-the-wrong-one-at-213am/

[11] Prompting For Refactoring Vs New Features - blog.vibecoder.me - https://blog.vibecoder.me/prompting-for-refactoring-vs-new-features

[12] [blog] Recurring Nightmares Software Anti Patterns In The Ai Era Te - robtyrie.medium.com - https://robtyrie.medium.com/recurring-nightmares-software-anti-patterns-in-the-ai-era-techs-d%C3%A9j%C3%A0-vu-a25dd351ada7

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

[14] [peer-reviewed] mdpi.com - AUTHORS UNAVAILABLE - https://www.mdpi.com/2079-8954/14/5/593

[15] [blog] Build Audit Trails Ai Coding Agents - mintmcp.com - https://www.mintmcp.com/blog/build-audit-trails-ai-coding-agents

[16] Understanding Code Provenance - fortegrp.com - https://fortegrp.com/insights/understanding-code-provenance

[17] [peer-reviewed] Operationalising artificial intelligence bills of materials for verifiable AI provenance and lifecycle assurance - Authors: Radanliev, Petar; Santos, Omar; Maple, Carsten; Atefi, Kayvan - Journal: Frontiers in Computer Science - https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2026.1735919/full

[18] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2604.23338v1

[19] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2605.12535v3

[20] [blog] Decoding Llm Outputs A Beginners Guide To Sampling Strategie - medium.com - https://medium.com/@xiaxiami/decoding-llm-outputs-a-beginners-guide-to-sampling-strategies-e0fa8d616924

[21] [blog] Llm Sampling - kachkach.com - https://kachkach.com/blog/llm-sampling

[22] File - repository.tudelft.nl - https://repository.tudelft.nl/file/File_44102dea-5f85-4261-92a9-ba444ca456a4?preview=1

[23] Multi Turn Semantic Drift - arize.com - https://arize.com/glossary/multi-turn-semantic-drift/

[24] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2603.11768v2

[25] [peer-reviewed] 2026.Findings Acl.57 - aclanthology.org - AUTHORS UNAVAILABLE - https://aclanthology.org/2026.findings-acl.57.pdf

[26] [preprint] An Empirical Study on the Potential of LLMs in Automated ... - AUTHORS UNAVAILABLE - https://arxiv.org/html/2411.04444v1

[27] ai-driven software development: enhancing code quality and - https://ijrdst.org/public/uploads/paper/252521740757533.pdf

[28] [peer-reviewed] Refactoring Loops in the Era of LLMs: A Comprehensive ... - AUTHORS UNAVAILABLE - https://www.mdpi.com/1999-5903/17/9/418

[29] [peer-reviewed] (PDF) Refactoring with LLMs: Bridging Human Expertise ... - AUTHORS UNAVAILABLE - https://www.researchgate.net/publication/396251015_Refactoring_with_LLMs_Bridging_Human_Expertise_and_Machine_Understanding

[30] [blog] Understanding LLM Failure Modes (And Why They Matter ... - https://medium.com/@RamPrakashD/understanding-llm-failure-modes-and-why-they-matter-more-than-the-model-itself-2104edccf3cd

[31] [blog] The Real Cost of AI-Generated Code Drift, and How to Stop It - https://www.mindstudio.ai/blog/ai-generated-code-drift-cost-analysis