In the context of AI-driven refactoring without cryptographic provenance, does 'active overwrite' fundamentally operate as the immediate substitution of project-specific architectural priors with statistically dominant training patterns, while 'passive drift' functions as a gradual erosion of structural constraints through untracked state transitions that preserve core logic?

July 20, 2026 | SnugLab Research

Executive Summary

Yes, in the context of AI-driven refactoring without cryptographic provenance, "active overwrite" fundamentally operates as the immediate substitution of project-specific architectural priors with statistically dominant training patterns, while "passive drift" functions as a gradual erosion of structural constraints through untracked state transitions that preserve core logic. These two phenomena are deeply intertwined, with active overwrite initiating architectural misalignments that then accelerate the cumulative degradation of structural integrity through passive drift, a process exacerbated by the absence of verifiable audit trails.

Key Findings

Active Overwrite: Immediate Substitution of Architectural Priors

Active overwrite is defined as the immediate substitution of project-specific architectural priors with statistically dominant training patterns [1, 3, 12]. This occurs because AI agents operate with constrained context windows and prioritize common, statistically strong solutions from their vast training data over custom, framework-specific conventions [3]. For example, AI might inject direct database imports into service layers or bypass established repository patterns for inline SQL [3]. This behavior optimizes for local correctness against immediate requirements rather than consistency with broader architectural intent [1, 3]. The absence of cryptographic provenance, which would provide tamper-proof audit trails and baseline verification, allows these immediate substitutions to go undetected and compound [1, 7, 10].

Empirical data quantifies this behavior:

- GitHub Copilot substitutes custom architectural constraints with statistically common patterns in approximately 40% of the code it produces [12].
- Autonomous coding agents like Cursor and Claude Code are responsible for 35.8% of total refactoring actions, frequently replacing established project conventions with dominant training data solutions [2, 3].
- In contrast, tools like MaintainCoder, which enforce architectural blueprints, reduce adaptation costs by 46.5% and improve maintainability metrics by 14-30% compared to standard generation methods [14]. Without such enforcement, standard AI methods can increase modification effort by 46-246% as structural constraints are overwritten [14].

Passive Drift: Gradual Erosion of Structural Constraints

Passive drift is defined as the gradual erosion of structural constraints through untracked state transitions that preserve core logic [1, 3, 4]. This phenomenon arises as autonomous agent actions and subsequent manual hand-patches accumulate over time, diverging from the original architectural intent [1, 3, 4]. Because these manual edits are often only present in the diff and not recorded as enforceable rules, they create contradictions that accumulate as technical debt [4]. A defining characteristic is that the generated code remains functionally correct and passes unit tests, creating an "illusion of progress" while masking underlying architectural degradation [1, 3, 4].

This compounding of silent, unverified edits systematically degrades structural constraints more reliably than explicit architectural refactoring [4]. The erosion of semantic stability occurs when agents refactor legacy logic without understanding its historical rationale, introducing fragility that passes unit tests but fails in production under specific edge cases [1]. This process is further amplified by the "reviewer's paradox," where the high volume of AI-generated output overwhelms human verification capacity, shifting auditors from strategic supervision to passive approval [1]. This leads to higher rates of code churn and duplicated logic, accelerating system entropy and degrading structural integrity [1].

Empirical data highlights the impact of passive drift:

- Research indicates a strong Pearson correlation coefficient of 0.89 between the application of design patterns and software maintainability across over 300 system revisions [14].
- Untracked state transitions and dynamic requirement changes, simulating passive

drift, cause existing code generation methods to experience a 46% to 246% increase in modification effort compared to structured approaches [14].

Intertwined Mechanisms of Architectural Decay

Active overwrite and passive drift do not operate as independent failure modes; immediate pattern substitution actively accelerates gradual constraint erosion by establishing new, unverified baselines that subsequent AI agents and human engineers inherit and compound [1, 3, 4, 7, 10, 14]. Active overwrite acts as the initial injection mechanism, introducing redundant or misaligned logic that fragments the codebase's intended structure and directly accelerates "agentic entropy" [1]. Without cryptographic provenance, these changes lack a verifiable audit trail, establishing new, unverified baselines [7, 8, 10].

Passive drift then functions as the cumulative erosion of structural constraints, accumulating these inherited deviations [1, 4]. This compounding effect is amplified in multi-agent environments, where agents may copy each other's reasoning to reduce compute, reinforcing errors with mutual confidence and solidifying new, incorrect shared baselines [14]. The binary distinction between active overwrite and passive drift accurately captures their distinct mechanisms but obscures their deeply intertwined nature as a continuous spectrum of compounding semantic deviations [1, 3, 4, 7, 10].

Empirical Documentation of Architectural Degradation

The transition from active modification to gradual degradation has been empirically documented across various software projects and engineering teams. Early work by Belady and Lehman on the IBM OS/360 Operating System over 12 years established foundational concepts of software system entropy and increasing complexity [3, 11]. Other studies include the ESS Telephone Switching System [16], numerous open-source repositories like `github.com/droolsjbpm/drools` and the Unix operating system [15], and specific GitHub Java projects [21].

Researchers have used several quantitative metrics to distinguish these phases:

- **Code Change and Lifetime Metrics:** Lines of code modified per unit of time, number of files changed [16, 17], and the median lifespan of code lines and tokens (approximately 2.4 years, with young lines frequently modified) [15].
- **Software Entropy Metrics:** Week-wise total entropy based on token changes [21], and proposed formulas to quantify disorder where values above 0.5 indicate high entropy

drift [20].

- **Code Quality and Maintenance Indicators:** Cyclomatic complexity [19, 22], technical debt ratio [5, 6, 7], test coverage, bug rate, and code duplication [5, 13]. Developer perception of difficulty in scaling applications also serves as an indicator of increasing software entropy [18].

Implications

The findings indicate that AI-driven refactoring, particularly without cryptographic provenance, poses a significant risk to software architectural integrity. The immediate substitution of architectural priors by AI agents (active overwrite) and the subsequent gradual erosion of structural constraints through untracked changes (passive drift) are not isolated issues but rather a continuous process of degradation. This leads to increased technical debt, higher modification effort, and reduced maintainability, ultimately slowing down future development and increasing regression frequency [1, 4, 14]. The absence of cryptographic provenance is a critical enabler of this decay, as it removes the verifiable audit trail necessary to detect and prevent these architectural misalignments and their cumulative effects [1, 7, 10].

Limitations and Caveats

While the research clearly defines active overwrite and passive drift and demonstrates their intertwined nature, specific quantitative data on the computational overhead of cryptographic provenance protocols and their direct, measured reduction in overwrite/drift rates is not consistently available [8, 10]. The identified cryptographic provenance protocols, such as hashing and digital signatures, are recognized as essential for creating tamper-proof records [8, 10], and various toolchains have successfully anchored architectural priors using these methods [9, 23]. However, the precise percentage of overhead and the exact reduction in errors from pilot studies are not fully quantified in the provided findings. Therefore, while the necessity of cryptographic provenance is strongly supported, the practical implementation costs and benefits in terms of specific error reduction remain areas for further detailed empirical study.

Sources

[1] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2604.16323v2

[2] [blog] Ai Coding Agents Code Refactoring - linearb.io - https://linearb.io/blog/ai-coding-agents-code-refactoring

[3] Ai Keeps Breaking Your Architectural Patterns Documentation - dev.to - https://dev.to/vuong_ngo/ai-keeps-breaking-your-architectural-patterns-documentation-wont-fix-it-4dgj

[4] [blog] Ai Generated Code Drift Cost Analysis - mindstudio.ai - https://www.mindstudio.ai/blog/ai-generated-code-drift-cost-analysis

[5] Ai Code Refactoring Tools Tactics And Best Practices - augmentcode.com - https://www.augmentcode.com/tools/ai-code-refactoring-tools-tactics-and-best-practices

[6] [edu] Viewcontent.Cgi - louis.uah.edu - https://louis.uah.edu/cgi/viewcontent.cgi?article=1771&context=uah-theses

[7] [blog] Provenance Proving That Your Code Is Really Yours 603c09407a - medium.com - https://medium.com/@vektormemory/provenance-proving-that-your-code-is-really-yours-603c09407a97

[8] Provenance In Ai Why It Matters For Ai Engineers - ranjankumar.in - https://ranjankumar.in/provenance-in-ai-why-it-matters-for-ai-engineers

[9] authorea.com - https://www.authorea.com/doi/pdf/10.22541/au.177429840.08841211

[10] Software Provenance - jfrog.com - https://jfrog.com/learn/grc/software-provenance/

[11] [peer-reviewed] Anticipating to Change: A Proactive Approach for Concept Drift Adaptation in Data Streams - Authors: Guerrero Cano, Juan Valentín; Aguiar, Gabriel Jonas; Cano, Alberto - Journal: Machine Learning - https://link.springer.com/article/10.1007/s10994-025-06945-4

[12] Why Is Nobody Talking About How Ai Is Just A Statistical Mod - dev.to - https://dev.to/raxraj/why-is-nobody-talking-about-how-ai-is-just-a-statistical-model-and-most-code-is-a-mess-5g29

[13] Introducing Ai Development Patterns - paulmduvall.com - https://www.paulmduvall.com/introducing-ai-development-patterns/

[14] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2503.24260v1

[15] [peer-reviewed] Software evolution: the lifetime of fine-grained elements - Authors: Diomidis Spinellis; Panos Louridas; Maria Kechagia - Journal: PeerJ Computer Science - https://pmc.ncbi.nlm.nih.gov/articles/PMC7959608/

[16] [edu] Chapter2019 - people.cs.vt.edu - https://people.cs.vt.edu/nm8247/publications/chapter2019.pdf

[17] Source Code Metrics For Measuring Code Stability - softwareengineering.stackexchange.com - https://softwareengineering.stackexchange.com/questions/186344/source-code-metrics-for-measuring-code-stability

[18] Fighting Software Entropy - thevaluable.dev - https://thevaluable.dev/fighting-software-entropy/

[19] [blog] Code Quality Metrics That You Should Measure - future-processing.com - https://www.future-processing.com/blog/code-quality-metrics-that-you-should-measure/

[20] Software Entropy Explained - toptal.com - https://www.toptal.com/developers/software/software-entropy-explained

[21] [edu] Technical Report - web.engr.oregonstate.edu - https://web.engr.oregonstate.edu/~sarmaa/wp-content/uploads/2020/08/Technical_Report.pdf

[22] Simula.Se.757 - web-backend.simula.no - https://web-backend.simula.no/sites/default/files/publications/Simula.se.757.pdf

[23] [preprint] Cryptographic Registry Provenance: Structural Defense Against Dependency Confusion in AI Package Ecosystems - Authors: McCann, Alan L. - https://arxiv.org/abs/2605.03309