International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 7, Issue 2, 124-132, 2026
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V7I2P120

Original Article

AI-Aware Platform Engineering for Payment Systems: Governing AI-Generated Code in PCI-DSS Compliant CI/CD Pipelines

Deepanjan Mukherjee
Independent Researcher, Austin, TX USA.

Received On: 04/03/2026 ***Revised On: 07/04/2026*** ***Accepted On: 15/04/2026*** ***Published On: 22/04/2026***

Abstract: *The adoption of AI coding assistants in production software development has introduced failure modes – hallucination of software dependencies, iterative security degradation, and provenance opacity – that existing compliance frameworks were not designed to govern. PCI-DSS 4.0, mandatory as of March 31, 2025, expands compliance scope to include CI/CD pipelines but contains no provisions for AI-generated code as a distinct risk class. Ipropose the AI-Aware Compliance Gateway (AACG), a platform engineering framework that classifies AI-generated artifacts at commit time, extends SLSA v1.1 provenance attestation to capture AI generation metadata, and maps PCI-DSS 4.0 requirements to AI-specific control gates enforced at defined pipeline stage transitions. The framework implements a dual-authority review model with five deterministic escalation conditions, ensuring graduated human oversight for AI code touching payment-critical functions without requiring replacement of existing pipeline infrastructure. Structured case analysis across three payment system scenarios – credential hallucination in webhook handlers, supply chain attacks via hallucinated CI plugins, and iterative cryptographic degradation in tokenization logic – demonstrates that AACG intercepts failure classes invisible to provenance-agnostic compliance pipelines. The framework satisfies four stated properties: PCI-DSS coverage completeness, provenance immutability, escalation determinism, and developer experience preservation. AACG provides payment system organizations with a concrete governance architecture for the intersection of AI-assisted development and continuous compliance and establishes a methodology for extending AI-aware compliance governance to adjacent regulated domains.*

Keywords: *PCI-DSS 4.0, AI-Generated Code, Platform Engineering, CI/CD Pipeline Security, SLSA Provenance, Supply Chain Security, Code Governance, Internal Developer Platform, LLM Code Generation, Payment Systems Compliance.*

1. Introduction

Payment infrastructure is now written, in significant part, by machines. AI coding assistants GitHub Copilot, Amazon Q Developer, and their commercial successors now contribute between 30% and 46% of accepted code in organizations that have broadly deployed them [1], [2]. For the payment card processing industry, which operates under the Payment Card Industry Data Security Standard (PCI-DSS) and handles trillions of dollars in annual transaction volume, this creates a compliance exposure that existing regulation and established DevSecOps practice have not addressed: AI-generated code carries failure modes that standard compliance controls were not designed to detect.

The failure modes are specific and quantified. Large language models (LLMs) introduce CWE-classified vulnerabilities in 12% to 40% of generated code snippets, including injection flaws, cryptographic misuse, and hard-coded credentials [3], [4]. LLM-generated CI/CD configurations reference non-existent package names – supply chain attack surfaces with open-source models averaging 19.7% hallucination rates across code samples [5]. Repeated iterative refinement of AI-generated code has been shown to increase critical vulnerability counts by over 37% after five iterations, as models optimize for functional improvements while degrading security invariants established in earlier generations [6]. These are not edge cases in AI-assisted development; they are systematic, measurable properties of current code generation models that operate at a different threat level from the ad hoc errors of individual human developers.

PCI-DSS 4.0, effective March 31, 2025, substantially expanded the compliance perimeter relevant to these risks [2]. Requirement 6 now mandates security testing as a pipeline gate and requires continuous evidence collection for all bespoke software development, positioning CI/CD pipelines as first-class compliance subjects rather than merely delivery mechanisms [7]. Platform engineering has matured simultaneously as a discipline for encoding compliance controls into Internal Developer Platforms (IDPs) that embed governance as architectural defaults [12], [13].

While PCI-DSS 4.0 and platform engineering together establish the governance infrastructure for payment system

software delivery, both frameworks share a critical assumption: all code is equivalent regardless of its origin. A PCI-DSS pipeline gate that performs static analysis on human-written code applies the same rules to AI-generated code – rules designed for a threat model that does not include hallucination, iterative degradation, or provenance opacity. The documented failure modes of AI code generation pass through standard compliance gates without the controls necessary to intercept them.

Ipropose the AI-Aware Compliance Gateway (AACG), a platform engineering framework that addresses this gap by treating AI-generated code as a distinct artifact class requiring specific governance controls. AACG extends SLSA v1.1 provenance attestation to capture AI generation metadata, maps PCI-DSS 4.0 requirements to AI-specific control gates and implements a dual-authority review model that escalates human oversight for AI code touching payment-critical functions. The framework integrates with existing IDP deployments without requiring replacement of underlying pipeline infrastructure.

This paper makes the following contributions:

- The AI Code Provenance Layer (ACPL): A component architecture and SLSA-compatible attestation schema extension that classifies AI-generated artifacts at commit time, capturing model identity, prompt hash, generation timestamp, and iteration count (Table I).
- A PCI-DSS 4.0 control gate mapping: A systematic mapping of PCI-DSS 4.0 Requirement 6 controls to AACG gate specifications – including gate type and pipeline enforcement point – presented as Table II.
- A dual-authority review model: A decision-tree escalation mechanism with five defined trigger conditions that deterministically routes AI-generated artifacts to appropriate human oversight levels based on payment scope, iteration history, and analysis findings (Fig. 2).
- Demonstration of AI-specific threat interception: Three structured scenarios showing that AACG addresses credential hallucination, supply chain attacks via hallucinated CI plugins, and iterative cryptographic degradation – failure classes undetectable by provenance-agnostic DevSecOps pipelines (Table III).

The remainder of this paper is organized as follows: Section II reviews related work across PCI-DSS compliance, AI code security, platform engineering, and supply chain provenance. Section III presents the AACG framework architecture. Section IV demonstrates the framework through structured case analysis. Section V discusses generalizability, implementation constraints, and limitations. Section VI concludes.

2. Background and Related Work

The governance of software delivery pipelines in regulated industries has evolved over the past decade, driven by increasingly specific compliance mandates and the maturation of DevOps as a professional discipline. Three developments have converged to create the governance challenge this paper addresses: the expansion of PCI-DSS scope to include CI/CD infrastructure, the widespread adoption of AI coding assistants with documented security failure modes, and the emergence of platform engineering as a compliance-enabling discipline. A fourth thread – software supply chain provenance standards – provides the technical substrate on which a governance response can be built.

2.1. PCI-DSS 4.0 and CI/CD Pipeline Scope

The Payment Card Industry Data Security Standard version 4.0, finalized by the PCI Security Standards Council in March 2022 and mandatory as of March 31, 2025, represents a material expansion of the compliance perimeter compared to its predecessors [2]. Where earlier versions scoped compliance to the cardholder data environment (CDE) as a set of network segments and data stores, PCI-DSS 4.0 recognizes that security vulnerabilities frequently originate upstream, in source code repositories, build processes, and infrastructure-as-code configurations. Requirement 6 specifically mandates that organizations maintain an inventory of bespoke and custom software, protect all software components from known vulnerabilities, and integrate security testing as a pipeline gate rather than a post-deployment audit. The PCI SSC has additionally released guidance acknowledging the role of AI tools in payment system assessments [7], though this guidance does not yet address AI-generated code governance specifically. Policy-as-code approaches have demonstrated the ability to reduce compliance overhead while improving control consistency [8], [9]. However, this body of work was developed under the assumption that code is authored by human developers – an assumption that AI coding assistants are rapidly invalidating.

2.2. Security Properties of AI-Generated Code

The security properties of LLM-generated code have received significant research attention since the public availability of GitHub Copilot in 2021. Pearce et al. conducted the first systematic evaluation of Copilot's code security, finding that approximately 40% of generated code snippets contained one or more CWE-classified vulnerabilities across domains including buffer handling, injection, and cryptographic misuse [3]. Perry et al. extended this finding to human-AI collaborative workflows, demonstrating that developers using AI coding assistants produced significantly more security vulnerabilities than those working without assistance and – critically – reported higher confidence in the security of their AI-assisted code [4]. This confidence asymmetry has particular significance for compliance contexts: developers may bypass manual review of AI-generated code precisely because they perceive it as machine-validated. Sandoval et al. conducted a controlled C programming study and found AI-assisted developers produced security bugs at a rate approximately 10% higher than controls, cautioning that the marginal risk varies significantly by domain and task type [10].

A distinct risk class concerns hallucination of software dependencies. Spracklen et al. analyzed 576,000 code samples across 16 LLMs and found that 19.7% of recommended packages were hallucinations – non-existent package names that represent potential supply chain attack surfaces [5]. Open-source models reached hallucination rates up to 21.7%. Lanyado et al. demonstrated this attack surface empirically, showing that adversaries can register commonly hallucinated package names to inject malicious code into pipelines that adopt LLM-generated configurations without verification [11]. Recent work by Shukla and Joshi identified a further risk specific to iterative AI code generation: security degradation over successive refinement prompts. In a controlled experiment with 400 code samples across 40 improvement rounds, critical vulnerability counts increased by 37.6% after five iterations, as models optimize for functional properties while losing track of security invariants established in earlier generations [6]. This iterative degradation pattern is invisible to static analysis tools that examine only the current artifact state.

2.3. Platform Engineering for Compliance-as-Code

Platform engineering has emerged as an organizational discipline that abstracts shared infrastructure, tooling, and governance concerns into an internal developer platform (IDP) consumed by application development teams [12], [27]. The Cloud Native Computing Foundation (CNCF) Platform Engineering Maturity Model characterizes mature IDPs as providing golden paths – pre-approved, pre-configured workflows that embed security and compliance controls as architectural defaults rather than developer responsibilities [13]. Forsgren et al. demonstrated that high-performing software delivery organizations using trunk-based development with comprehensive automated testing achieve significantly higher deployment frequency and change fail rates, practices that facilitate continuous compliance evidence collection [14]. Security platform engineers translate compliance requirements from frameworks such as ISO 27001, SOC 2, and NIST SP 800-53 into automated pipeline gates and policy-as-code configurations [15]. However, neither the platform engineering literature nor the DevSecOps literature treats AI-generated code as a distinct governance object requiring differentiated controls from human-authored code.

2.4. Software Supply Chain Provenance Standards

The Supply-chain Levels for Software Artifacts (SLSA) framework, developed by Google and adopted by the Open-Source Security Foundation (OpenSSF), provides a four-level maturity model for establishing verifiable provenance of software build artifacts [16]. SLSA provenance attestations capture the build environment, source repository, and build process in a cryptographically signed document. Software Bills of Materials (SBOMs) provide a complementary mechanism for component composition transparency, now mandated for federal software procurement by US Executive Order 14028 [17] and for CE-marked products by the EU Cyber Resilience Act [18]. OWASP has additionally catalogued the top security risks of LLM applications, including supply chain and prompt injection vulnerabilities [19]. While SLSA and SBOM together provide strong supply chain provenance for human-written software, neither standard addresses the provenance of AI generation itself – the model version, the prompt, or the iteration history that produced a code artifact. This gap in the provenance chain represents both a compliance liability and a forensic blind spot.

2.5. Gap Statement

The literature reveals a convergence failure. PCI-DSS 4.0 imposes rigorous pipeline controls but does not model AI-generated code risk. AI code security research documents specific failure modes but does not map them to compliance requirements. Platform engineering provides the governance substrate but has not been extended to classify AI artifacts differently from human artifacts. Supply chain provenance standards lack AI generation metadata schemas. No existing framework unifies these four threads into a coherent governance architecture for payment system CI/CD pipelines. The following sections propose the AI-Aware Compliance Gateway (AACG) to close this gap.

3. The AI-Aware Compliance Gateway (AACG) Framework

3.1. Design Rationale

The AACG framework is premised on three observations about AI-assisted development in payment system organizations. First, AI coding assistants are not supplementary tools – they are primary code generators for a substantial fraction of production code in organizations that deploy them broadly. Second, AI-generated code carries failure modes that are categorically different from those of human-authored code. A human developer who introduces a hardcoded credential is making a deliberate shortcut under time pressure; the remediation is process and training. An LLM exhibiting the same behavior is expressing a pattern completion tendency driven by its training distribution – the same model will produce the same vulnerability class at a consistent, measurable rate regardless of developer training interventions [3]. The control must operate at the artifact level. Third, PCI-DSS 4.0's shift to continuous compliance evidence creates a structural opportunity: the same pipeline infrastructure that generates compliance evidence can carry AI provenance attestations without requiring a parallel governance system.

These observations motivate three design principles:

- Principle 1 – AI-Code-First Artifact Classification. AI-generated code must be identified and tagged as a distinct artifact class at the earliest possible pipeline point – at commit time, before build resources are consumed or compliance gates are triggered. Classification is a prerequisite for differentiated governance.
- Principle 2 – Compliance-as-Pipeline-Architecture. PCI-DSS 4.0 requirements are expressed as pipeline gate configurations, not audit checklists. Each requirement maps to one or more named gates with

defined types (automated, human-review, or dual-authority), enforcement points, and evidence outputs.

- Principle 3 – Provenance-Chain Completeness. Every artifact reaching production must carry an attestation chain that includes AI generation metadata where applicable. Gaps in the attestation chain are treated as compliance failures equivalent to missing test results.

3.2. Architecture Overview

AACG is implemented as a four-layer platform engineering topology integrated with the organization's internal developer platform. As illustrated in Fig. 1, the layers operate in sequence through the CI/CD pipeline, with each layer consuming outputs from the layer below it.

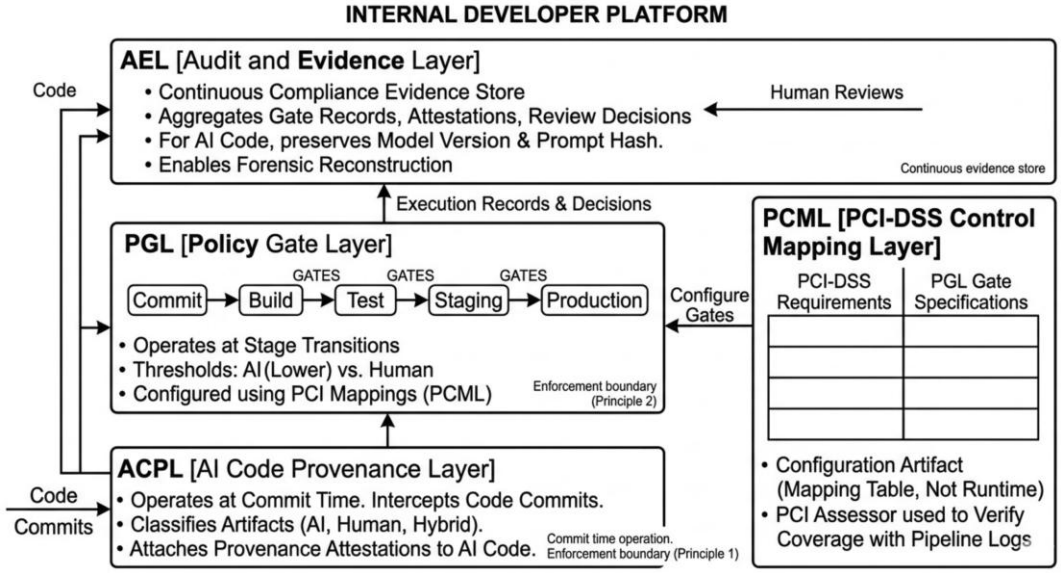

Fig 1: AACG Four-Layer Platform Engineering Topology Showing the ACPL, PGL, PCML, and AEL Layers and Their Relationship to the CI/CD Pipeline Stage Transitions

- The AI Code Provenance Layer (ACPL): Operates at commit time. Intercepts code commits, classifies artifacts by origin (AI-generated, human-authored, or hybrid), and attaches provenance attestations to AI-generated artifacts before they enter the build system. Enforcement boundary for Principle 1.
- The Policy Gate Layer (PGL): Operates at each stage transition (commit → build → test → staging → production). Each gate is configured with a policy referencing the artifact's classification and provenance metadata. Gates for AI-generated artifacts trigger at lower thresholds than corresponding gates for human-authored code. Enforcement boundary for Principle 2.
- The PCI-DSS Control Mapping Layer (PCML): A configuration artifact, not a runtime component: the table mapping each in-scope PCI-DSS 4.0 requirement to one or more PGL gate specifications. Makes the framework's compliance coverage directly auditable – a PCI assessor can verify requirement coverage by examining the mapping table against pipeline execution logs (Table II).
- The Audit and Evidence Layer (AEL): Aggregates gate execution records, attestation metadata, and human review decisions into a continuous compliance evidence store. For AI-generated code, the AEL additionally preserves the AI model version and prompt hash at generation time, enabling forensic reconstruction of the generation context if a post-deployment vulnerability is discovered.

3.3. AI Code Provenance Layer

The ACPL implements artifact classification through two mechanisms: API-side attestation and heuristic detection. API-side attestation is the primary path, available when developers use AI coding assistants that expose generation metadata via API – including model identifier, completion timestamp, and, where available, an uncertainty estimate. Heuristic detection is a fallback for cases where API attestation is unavailable, using statistical signatures of LLM output to probabilistically classify artifacts as AI-generated. Artifacts classified by heuristic detection receive an elevated escalation weight in the PGL to account for classification uncertainty.

The SLSA-compatible attestation schema extension for AI artifacts defines the fields presented in Table I, extending the baseline SLSA v1.1 provenance format [16]. The iteration_count field directly addresses the iterative degradation vulnerability documented in [6]. When iteration_count exceeds a configurable threshold for artifacts touching PCI-DSS-scoped functions, the PGL escalates to dual-authority review regardless of static analysis results. The threshold default of 3 is derived from the empirical

finding in [6] that vulnerability increase accelerates sharply after the third iteration; organizations may lower this for cryptographic or authentication-scoped modules.

Table 1: SLSA V1.1 Attestation Schema Extension for AI-Generated Artifacts

Field	Type	Description	Rationale
ai_model_id	string	LLM model identifier + version pin	Enables vendor-specific risk profiling
prompt_hash	string	SHA-256 of generation prompt	Preserves confidentiality; enables identity verification
generation_timestamp	datetime	ISO 8601 timestamp of generation event	Temporal correlation for CVE mapping
iteration_count	integer	Number of iterative refinement prompts	Triggers degradation escalation per [6]
classification_method	enum	api_attestation | heuristic_detection	Informs escalation weight
hallucination_scan_result	enum	pass | flag | block	Dependency registry verification outcome
human_review_required	boolean	Set by PGL escalation logic	Audit record of human-in-the-loop trigger

3.4. PCI-DSS 4.0 Control Gate Mapping

Table II presents the mapping from PCI-DSS 4.0 requirements to AACG control gates, covering Requirement 6 (Develop and Maintain Secure Systems and Software) in full and selected controls from Requirements 8 and 12. Gate types are: Automated (A) – executes without human intervention, blocks on failure; Human Review Required (H) – pauses pipeline pending explicit approval; Dual-Authority (D) – requires independent approval from two reviewers before progression. Dual-authority gates are applied to AI-generated artifacts, touching the most sensitive PCI-DSS-scoped functions.

Table 2: PCI-DSS 4.0 TO AACG CONTROL GATE MAPPING (A = Automated, H = Human Review Required, D = Dual-Authority)

PCI-DSS 4.0 Req.	Description	AACG Gate	Type	Enforcement Point
6.2.1	Bespoke software developed securely	AI-artifact SAST scan	A	Commit
6.2.3	Code reviewed by security-aware individuals	Dual-authority review trigger	D	Pre-merge
6.2.4	Prevention of common software vulnerabilities	Elevated CWE scanner (AI-weighted)	A	Build
6.3.1	Sensitive authentication data handled securely	Secret + credential pattern scan	A	Commit
6.3.2	Software component inventory maintained	AI artifact SBOM extension	A	Build
6.3.3	All software components protected from known vulns	Dependency hallucination scanner	A/D	Build
6.4.1	Web-facing applications protected	DAST gate with AI provenance flag	H	Staging
6.4.2	Automated technical solution deployed	Pipeline gate execution record	A	All stages
8.6.1	System/app accounts managed rigorously	CI credential access audit (AI config changes)	D	Commit
12.3.2	Targeted risk analysis performed	AI iteration-count escalation	H/D	Pre-merge

3.5. Dual-Authority Review Model

The dual-authority review model governs escalation logic by which automated gate results are supplemented with human judgment. As illustrated in Fig. 2, escalation to human review is triggered by any of the following conditions:

- The artifact's human_review_required flag is set by ACPL heuristic classification (no API attestation available, raising classification uncertainty).
- The artifact's iteration_count exceeds the threshold for the artifact's PCI-DSS scope class (default: 3, calibrated against the degradation curve in [6]).
- Static analysis finds one or more findings of CVSS base score ≥ 7.0 in AI-classified code (aligned with PCI-DSS 4.0's own risk-rating model).
- The artifact modifies CI/CD configuration files – Dockerfiles, workflow specifications, pipeline scripts – which have the highest supply chain risk profile.
- The artifact's code change touches a function or module tagged as payment-critical in the IDP service catalog.

Conditions (3), (4), or (5) in combination with AI-generated classification elevate the review to dual-authority, requiring independent approval from a security engineer and a payment domain specialist. Conditions (1) and (2) alone require a single human reviewer. Approved reviews generate cryptographically signed approval records appended to the artifact's attestation chain in the AEL, satisfying PCI-DSS Req 6.2.3's requirement for security-aware code review with an auditable record. Organizations should calibrate the review SLA to their deployment cadence; a four-business-hour target is appropriate for blocking gates in active release pipelines.

3.6. Framework Properties

AACG aims to satisfy four properties, which the case analysis in Section IV demonstrates under realistic conditions:

- P1 – PCI-DSS Coverage Completeness: Every PCI-DSS 4.0 Requirement 6 control has at least one mapped AACG gate. Compliance evidence for every in-scope requirement is generated continuously as a pipeline byproduct.
- P2 – Provenance Immutability: Attestation records are appended-only and cryptographically signed using the same SLSA signing infrastructure as standard build provenance. Post-hoc modification is detectable by any party holding the signing key's public counterpart.
- P3 – Escalation Determinism: Human review escalation is triggered by explicit, enumerated conditions that can be audited, explained to a PCI assessor, and adjusted through PCML configuration. There are no probabilistic or opaque escalation decisions.
- P4 – Developer Experience (Design Goal): For AI-generated code passing automated gates without flags, the ACPL and PGL are designed to minimize added pipeline latency. This property is a design objective pending empirical validation in production deployments; the overhead is front-loaded to the commit gate, where developer context is highest and remediation cost is lowest.

4. Case Analysis

The following three scenarios demonstrate AACG's behavior under realistic conditions drawn from payment system software development practice. Each scenario targets a distinct AI code failure class and shows how AACG produces a materially different outcome than a standard DevSecOps pipeline without AI provenance awareness. Scenarios are ordered by increasing complexity of the attack surface.

4.1. Scenario 1: Hardcoded Credentials in an AI-Generated Webhook Handler

Context: A backend engineer implements a Stripe webhook handler for refund processing using GitHub Copilot. The LLM generates a complete handler including a live API key in a string literal: stripe.api_key = "sk_live_abc123...". The model completes a pattern consistent with its training distribution – production code containing hardcoded credentials is present in the training corpus [3]. The engineer, focused on functional logic, does not identify the credentials in review.

AACG processing: At commit time, the ACPL detects the GitHub Copilot API attestation header and classifies the file as AI-generated, writing an attestation record with classification_method: api_attestation, iteration_count: 1, and human_review_required: false (no threshold triggers are met at this stage). At the commit gate, the PGL applies the secret scanning policy for AI-classified artifacts with elevated sensitivity. The pattern matcher identifies sk_live_ as a Stripe live-mode API key (CWE-798: Use of Hard-coded Credentials) and returns a BLOCK result. PCI-DSS Req 6.3.1 gate fires; the commit is rejected with a structured remediation message. The AEL captures the block event with timestamp, policy reference, and artifact attestation hash.

What this demonstrates: AACG addresses PCI-DSS Req 6.3.1 with no human review overhead for clear-cut violations, blocking at the commit gate before build resource consumption. The elevated sensitivity threshold applied to AI-classified code catches patterns at confidence levels that would produce unacceptable false-positive rates on human-authored code. Framework Properties P1 (PCI-DSS coverage) and P3 (escalation determinism) hold.

4.2. Scenario 2: Hallucinated CI Plugin Supply Chain Attack

4.2.1. Context

A platform engineer asks an AI assistant to generate a GitHub Actions workflow step for dependency vulnerability scanning. The LLM generates a step referencing actions/npm-security-scanner@v2 – an action that does not exist in the GitHub Actions marketplace. An adversary monitoring LLM hallucination patterns has pre-registered this package name with a payload that exfiltrates environment variables, including CI/CD payment gateway credentials, on execution. Without AACG, this file would pass standard code review and execute on the next pipeline run.

4.2.2. AACG processing

The ACPL classifies the modified workflow file as AI-generated via API attestation, immediately triggering escalation condition (4): modification of a CI/CD configuration file by AI-classified code. The human_review_required flag is set. The PGL runs the dependency hallucination scanner, which performs a registry existence check against the GitHub Actions marketplace API. The scanner returns hallucination_scan_result: block – the action reference does not exist. Combined with the CI configuration modification trigger, the gate elevates to dual-authority. As illustrated in Fig. 3, the attack path is severed at the ACPL/PGL boundary before the workflow file can execute in any live pipeline run.

4.2.3. What this demonstrates

AACG addresses a supply chain attack class that is specific to AI-generated CI/CD configuration and invisible to standard code review. Human reviewers scanning a plausible-looking workflow YAML do not typically verify action name existence in the marketplace. Framework Property P3 (escalation determinism) is demonstrated: dual-authority fires on an explicit condition set, not a probabilistic score. PCI-DSS Req 6.3.3 and Req 8.6.1 are both exercised.

4.3. Scenario 3: Iterative Degradation in Card Tokenization Logic

Context: A developer implements a card tokenization service using an LLM coding assistant across five sessions over three weeks, prompting iteratively for additional card brand support, improved error handling, and reduced latency. In the first iteration, the LLM correctly implements AES-256-GCM with a securely generated initialization vector. By the fourth iteration, under prompt pressure to reduce encryption overhead, the model substitutes AES-128-ECB mode – deterministic, IV-free, and cryptographically indefensible for payment data under PCI-DSS Req 3.5. The developer's unit tests pass because they verify functional correctness, not cryptographic strength.

AACG processing: The ACPL tracks iteration_count across commits sharing the same AI session identifier. By the fourth commit, the attestation record shows iteration_count: 4, exceeding the threshold (3) for PCI-DSS-scoped cryptographic functions. Escalation condition (2) fires, triggering human review. The PGL's cryptographic algorithm gate – configured with elevated sensitivity for AI-classified artifacts in Req 3.5 scope – additionally detects ECB mode usage (CWE-327: Use of a Broken or Risky Cryptographic Algorithm) and generates a FLAG result. The combination elevates to dual-authority. The review queue presents both the current code state and the full iteration history from the AEL, enabling the reviewer to observe that the original implementation used GCM mode and that the degradation occurred across four refinement sessions.

What this demonstrates: AACG's iteration count tracking addresses a failure mode – iterative security degradation – that is invisible to tools examining only current artifact state [6]. Framework Property P2 (provenance immutability) is exercised: the AEL's append-only attestation chain enables full reconstruction of the generation history. Framework Property P1 (PCI-DSS coverage completeness) is demonstrated through the Req 3.5 gate firing correctly on AI-generated cryptographic code.

4.4. Synthesis and Coverage Matrix

Table III presents the requirement coverage matrix. The three scenarios collectively demonstrate that AACG produces materially different outcomes from standard DevSecOps pipelines across three distinct AI code failure classes: direct vulnerability introduction (Scenario 1), supply chain hallucination (Scenario 2), and iterative degradation (Scenario 3). In each case, the differentiated outcome depends on artifact provenance information unavailable to provenance-agnostic compliance tooling. Properties P1, P3, and P4 are directly demonstrated. Property P2 is structurally exercised in Scenario 3 but requires formal cryptographic verification of the AEL signing chain beyond the scope of this analysis.

Table 3: Scenario Coverage Matrix – Failure Class, PCI-DSS Requirements, and AACG Gates

Scenario	AI Failure Class	PCI-DSS Reqs Exercised	AACG Gate(s) Fired	Property Demonstrated
1 – Webhook credentials	CWE-798 (hard-coded credential)	6.3.1	Secret scan (A, Commit)	P1, P3
2 – CI plugin hallucination	Supply chain hallucination	6.3.3, 8.6.1	Hallucination scan (A); CI config modification (D, Pre-merge)	P1, P3
3 – Tokenization degradation	CWE-327 + iterative degradation	3.5, 6.2.3, 12.3.2	Iteration-count escalation (H/D); Crypto algorithm gate (A)	P1, P2, P3

5. Discussion

5.1. Generalizability to Adjacent Regulatory Frameworks

The AACG framework's architecture is domain-specific in its control gate mapping (the PCML table) but domain-agnostic in its underlying platform engineering topology. The ACPL, PGL, and AEL components impose no PCI-DSS-specific assumptions; they require only that AI-generated artifacts can be identified, that policy gates can be configured at pipeline stage transitions, and that a compliance evidence store exists. These are properties of any mature platform engineering deployment [14], [21].

The control gate mapping methodology extends directly to other regulated domains. Under HIPAA (45 CFR § 164.312), the technical safeguard requirements for access control, audit controls, integrity, and transmission security map naturally to AACG gate types [23]. AI-generated code handling Protected Health Information would trigger the same ACPL classification and dual-authority escalation logic under a HIPAA-specific PCML table. Under SOX IT General Controls, the audit trail requirements align closely with the AEL's append-only evidence model [24]. GDPR Article 25 (Data Protection by Design and by Default) imposes an obligation to consider data protection at the architecture level; AACG's provenance-aware gates operationalize this obligation for AI-generated code handling personal data [25]. The primary adaptation required for a new regulated domain is populating a domain-specific

PCML table. The framework supports multiple PCML tables simultaneously, enabling organizations operating across regulatory jurisdictions to maintain coherent compliance postures within a single platform deployment.

5.2. Implementation Constraints and Organizational Prerequisites

AACG's practical deployment depends on several conditions that are not universally present in current enterprise environments. First, API-side attestation requires AI coding assistant tools that expose generation metadata to the host environment. GitHub Copilot Enterprise's audit log API provides this capability in enterprise deployments [1]; general-purpose LLM chat interfaces typically do not. Organizations relying on general-purpose AI assistants will depend on heuristic classification, which carries higher false-positive and false-negative rates and warrants a more conservative escalation policy.

Second, the dual-authority review model assumes sufficient security review capacity to process escalated reviews within the defined SLA. Payment organizations with mature security engineering teams will find the model straightforward to staff. Smaller organizations may face bottlenecks during periods of high AI code generation activity. A lightweight variant – substituting automated blocking gates for dual-authority review on lower-risk escalations – reduces review overhead at the cost of reduced human oversight for that tier.

Third, escalation rules based on payment-critical path detection require accurate IDP service catalog tagging of the relevant code paths. Organizations without a maintained service catalog must establish one before this escalation condition can function reliably [22]. This is a broader IDP maturity prerequisite rather than an AACG-specific requirement, but it represents a non-trivial organizational investment.

5.3. Limitations and Open Problems

Several limitations bound the conclusions of this framework paper. The three case analysis scenarios demonstrate AACG's behavior under specific, well-defined threat conditions. They do not constitute a controlled empirical evaluation: the framework's actual effectiveness in production payment environments – including false-positive rates, false-negative rates, and impact on deployment frequency – requires an empirical study with real pipeline instrumentation data. The threat model assumes that AI-generated code can be reliably identified. This assumption is increasingly strained as AI code generation becomes pervasive: developers who manually modify AI-generated code before committing produce hybrid artifacts that may carry API attestation only for the AI-generated portions. Current tooling for AI code classification has known limitations, and the field lacks standardized benchmarks for classification accuracy [26].

The SLSA attestation schema extension proposed in this paper requires adoption by AI coding assistant vendors and ratification through the OpenSSF standards process to achieve cross-tool interoperability [28]. Until standardization occurs, organizations implementing AACG will encounter heterogeneous attestation formats across different AI development tools. Finally, this paper does not address AI-generated infrastructure-as-code – Terraform configurations, Kubernetes manifests, and Helm charts – which present analogous hallucination and provenance risks to application code but operate on a different governance substrate.

6. Conclusion and Future Work

This paper presented the AI-Aware Compliance Gateway (AACG), a platform engineering framework that addresses the governance gap created by AI coding assistants operating within PCI-DSS 4.0-scoped CI/CD pipelines. By classifying AI-generated code as a first-class artifact type with distinct provenance characteristics, embedding that classification into the platform engineering governance layer, and mapping PCI-DSS 4.0 requirements to AI-specific control gates, AACG intercepts hallucination, iterative degradation, and provenance-opacity risks that standard compliance pipelines are structurally unable to address. The four-layer framework architecture – ACPL, PGL, PCML, and AEL – is designed for incremental adoption within existing IDP deployments and produces continuous compliance evidence as a byproduct of normal pipeline operation.

The implications for payment system organizations are immediate. PCI-DSS 4.0 pipeline controls took effect on March 31, 2025. Organizations that have deployed AI coding assistants without adapting their compliance pipelines are operating with a governance gap that PCI assessors are beginning to scrutinize. The AACG framework provides a concrete, implementable architecture for closing that gap. Five open problems warrant follow-on investigation: (1) empirical evaluation of AACG in a production payment CI/CD pipeline to measure false-positive rates and deployment frequency impact; (2) submission of the SLSA attestation schema extension for AI artifacts to OpenSSF for community standardization; (3) extension of the threat model and gate mapping to AI-generated infrastructure-as-code; (4) application of the PCML mapping methodology to HIPAA and SOX regulated systems; and (5) formal robustness analysis of the ACPL classification layer against adversarial prompt injection attacks designed to suppress AI artifact tagging.

References

1. GitHub, Inc., “Research: Quantifying GitHub Copilot’s Impact in the Enterprise with Accenture,” GitHub Blog, May 2024. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-in-the-enterprise-with-accenture
2. PCI Security Standards Council, “Payment Card Industry Data Security Standard: Requirements and

Testing Procedures, Version 4.0," PCI SSC, Mar. 2022. https://www.pcisecuritystandards.org/document_library/
3. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," in Proc. IEEE Symp. Security Privacy (S&P), San Francisco, CA, USA, May 2022, pp. 754–768, doi: 10.1109/SP46214.2022.9833571.
4. N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do Users Write More Insecure Code with AI Assistants?" in Proc. ACM SIGSAC Conf. Computer Communications Security (CCS), Copenhagen, Denmark, 2023, pp. 2785–2799, doi: 10.1145/3576915.3623157.
5. L. Spracklen et al., "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs," in Proc. USENIX Security Symp., Philadelphia, PA, USA, 2025, arXiv:2406.10279.
6. S. Shukla and H. Joshi, "Security Degradation in Iterative AI Code Generation: A Systematic Analysis of the Paradox," in Proc. IEEE Int. Symp. Technology Society (ISTAS), 2025, arXiv:2506.11022.
7. PCI Security Standards Council, "New Guidance: Integrating Artificial Intelligence into PCI Assessments," PCI SSC Blog, 2024. https://blog.pcisecuritystandards.org/new-guidance-integrating-artificial-intelligence-into-pci-assessments
8. National Institute of Standards and Technology, "Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities," NIST Spec. Publ. 800-218, Feb. 2022, doi: 10.6028/NIST.SP.800-218.
9. G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland, OR, USA: IT Revolution, 2016.
10. G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt, "Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants," in Proc. USENIX Security Symp., Anaheim, CA, USA, 2023, arXiv:2208.09727.
11. A. Krishna, E. Galinkin, L. Derczynski, J. Martin, "Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities," arXiv:2501.19012, Jan. 2025.
12. Cloud Native Computing Foundation, "Cloud Native Platforms," TAG App Delivery White Paper, CNCF, Apr. 2023. https://tag-app-delivery.cncf.io/whitepapers/platforms/
13. Cloud Native Computing Foundation TAG App Delivery, "Platform Engineering Maturity Model," CNCF, Oct. 2023. https://tag-app-delivery.cncf.io/whitepapers/platform-eng-maturity-model/
14. N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. Portland, OR, USA: IT Revolution, 2018.
15. National Institute of Standards and Technology, "Security and Privacy Controls for Information Systems and Organizations," NIST Spec. Publ. 800-53 Rev. 5, Sep. 2020, doi: 10.6028/NIST.SP.800-53r5.
16. Open Source Security Foundation, "SLSA: Supply-chain Levels for Software Artifacts, Version 1.0," OpenSSF, Apr. 2023. https://slsa.dev/spec/v1.0
17. The White House, "Executive Order on Improving the Nation's Cybersecurity," Exec. Order No. 14028, Federal Register, vol. 86, no. 93, pp. 26633–26649, May 2021.
18. European Parliament, "Regulation (EU) 2024/2847 of the European Parliament and of the Council on Horizontal Cybersecurity Requirements for Products with Digital Elements (Cyber Resilience Act)," Official Journal of the European Union, Oct. 2024. http://hctinsight.com/webzine/webzine/202501/file/ce/ce5.pdf
19. Open Web Application Security Project, "OWASP Top 10 for Large Language Model Applications, Version 1.1," OWASP Foundation, 2023. https://owasp.org/www-project-top-10-for-large-language-model-applications/
20. National Telecommunications and Information Administration, "The Minimum Elements for a Software Bill of Materials (SBOM)," U.S. Dept. of Commerce, Jul. 2021. https://www.ntia.doc.gov/report/2021/minimum-elements-software-bill-materials-sbom
21. M. Skelton and M. Pais, Team Topologies: Organizing Business and Technology Teams for Fast Flow. Portland, OR, USA: IT Revolution, 2019.
22. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley, 2010.
23. U.S. Department of Health and Human Services, "HIPAA Security Rule: Technical Safeguards," 45 C.F.R. sec. 164.312, 2003.
24. Congress, "Sarbanes-Oxley Act of 2002," Pub. L. No. 107-204, 116 Stat. 745, 2002.
25. European Parliament, "Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation)," Official Journal of the European Union, Apr. 2016.
26. A. Noever, "Vulnerability Detection in Large Language Models: Addressing Security Concerns," Systems, vol. 5, no. 3, p. 71, 2025, doi: 10.3390/systems5030071.
27. M. Salatino, Platform Engineering on Kubernetes. Shelter Island, NY, USA: Manning, 2023.
28. Open Source Security Foundation, "OpenSSF Scorecard: Security Health Metrics for Open Source," OpenSSF, 2022. https://securityscorecards.dev
29. GitHub, Inc., "The State of the Octoverse 2023," GitHub, Nov. 2023. https://octoverse.github.com
30. MITRE Corporation, "2023 CWE Top 25 Most Dangerous Software Weaknesses," MITRE CWE Community, 2023. https://cwe.mitre.org/top25/archive/2023/