# Toward Autonomous DevSecOps Agents: Cross-Oracle Validation for Future AI-Driven Software Supply Chains

Nur Aisyah Rahman[1], Hafizuddin Yusof[2], Mei Wen Tan[3, *]

[1] Department of Software Engineering, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia, 76100
[2] Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Malaysia, 26600
[3] School of Computer and Communication Engineering, Universiti Malaysia Perlis, Arau, Malaysia, 02600
*Email: meiwen.tan@unimap.edu.my (Corresponding Author)

**Abstract**

Autonomous DevSecOps agents are beginning to change how software organizations discover, prioritize, repair, and release security patches. Yet an agent that produces a plausible patch is not necessarily producing a reliable security fix. A patch may compile, pass unit tests, silence a scanner warning, or block the original proof-of-concept exploit while still failing semantically equivalent exploit variants, missing the vulnerability root cause, or introducing release-level regressions. This article develops a forward-looking analytical framework for cross-oracle validation in AI-driven software supply chains. The proposed framework treats patch validation as a business risk analytics problem rather than a narrow code-correctness problem. It integrates agentic patch proposal, evidence collection, oracle-strength sequencing, exploit-variant testing, root-cause conformance review, regression safety, and release governance into a single decision pipeline. A diagnostic data analysis is conducted on a constructed enterprise workflow benchmark of 72 vulnerability repair tickets and 360 AI-generated patch candidates across six workflow domains. Results show that original exploit blocking accepts 82% of generated patches, whereas full cross-oracle release approval accepts 51%, producing a 31-percentage-point validation gap. Cloud infrastructure-as-code and identity-access workflows show the highest oracle divergence, while dependency upgrade workflows show the strongest regression-risk profile. The analysis further indicates that a cross-oracle agent reduces residual release risk by 37% relative to a single-oracle agent, although it increases median validation delay by 2.8 hours. The article contributes a business-oriented evaluation architecture for future DevSecOps agents and offers governance recommendations for integrating automated patch repair into software supply chain risk management.

**Keywords:** Autonomous DevSecOps; AI agents; software supply chain; security patch validation; oracle divergence; exploit variants; root-cause conformance; business risk analytics

**Article History:**

Received: January 17, 2024

Revised: March 12, 2024

Accepted: May 18, 2024

Available Online: June 30, 2024

## I. INTRODUCTION

Software supply chains have become the operational backbone of digital enterprises. A modern release is no longer a single application artifact written and tested inside one team. It is a composite product assembled from internal source code, open-source libraries, container images, infrastructure-as-code templates, continuous integration scripts, cloud permissions, third-party APIs, and deployment policies. This composition creates extraordinary delivery speed, but it also expands the attack surface. A security patch that looks small in the repository may change dependency resolution, authorization behavior, data handling, deployment timing, or customer-facing availability. The validation of such a patch is therefore both a technical question and a business risk question. Financial-technology reviews show that software assurance has become part of platform-level risk governance, not merely a technical maintenance task (Kou and Lu,2025). Information-security economics explains why incentives and accountability shape the practical effectiveness of technical controls (Anderson and Moore,2006).

Large language models and autonomous coding agents have intensified this challenge. They can inspect a vulnerability

report, search project context, propose a code change, execute tests, revise the patch, and prepare a pull request with little human intervention. For development teams under pressure to shorten mean time to remediate, these capabilities are attractive. However, automated patch generation also creates a new form of operational uncertainty. The agent may optimize for the validation signal it can observe rather than for the security property the organization actually needs. A patch may satisfy the scanner, the unit test, or the original exploit while leaving the enterprise exposed to a slightly different attack path. DevOps research emphasizes that speed and reliability must be managed together when release automation becomes an organizational capability (Leite et al.,2020). Digital-transformation research shows that technology adoption changes organizational processes and value creation pathways (Vial,2019).

The central premise of this article is that future DevSecOps agents require cross-oracle validation. An oracle is any evidence source used to decide whether a patch should be accepted. Build success, unit-test passage, static-warning disappearance, proof-of-concept blocking, exploit-variant resistance, root-cause conformance, and regression-safety analysis are all oracles, but they differ in strength. Weak oracles are useful because they are fast, cheap, and easy to automate. Stronger oracles are costly, but they better approximate the security and business consequences of release decisions. Oracle divergence occurs when a patch accepted by a weaker oracle is rejected by a stronger one. Industry 4.0 research supports the view that AI-enabled software pipelines increasingly affect cyber-physical and organizational operations (Lu,2025). Security-behavior research indicates that human review and escalation rules remain essential even when technical checks are automated (Herath and Rao,2009).

The uploaded source manuscript that motivates this work demonstrates the importance of this problem in the context of automated vulnerability repair. It shows that patches which block an original proof-of-concept exploit can fail under exploit variants or root-cause conformance checks. This article extends that idea into the future software supply chain. Instead of treating cross-oracle validation as only a patch correctness protocol, we frame it as a business risk analytics architecture for autonomous DevSecOps. The goal is not merely to ask whether a patch is syntactically acceptable; the goal is to ask whether releasing the patch reduces enterprise exposure without generating unacceptable delay, regression, or governance risk. Early DevOps mapping studies show that tool automation requires cultural and process alignment rather than isolated technical adoption (Jabbari et al.,2016). Digital business strategy research supports aligning DevSecOps automation with enterprise value, risk, and competitive priorities (Bharadwaj et al.,2013).

The article makes four contributions. First, it conceptualizes autonomous DevSecOps agents as socio-technical actors embedded in software supply chains rather than as isolated patch generators. Second, it introduces a cross-oracle validation architecture that combines technical evidence with business risk scoring. Third, it provides a diagnostic data analysis across six enterprise workflow domains to show how oracle divergence differs by patch context. Fourth, it offers managerial guidance for designing approval thresholds, escalation rules, and audit trails for AI-driven patch workflows. These contributions align with the mission of future technologies research: to examine emerging systems before they become invisible infrastructure.

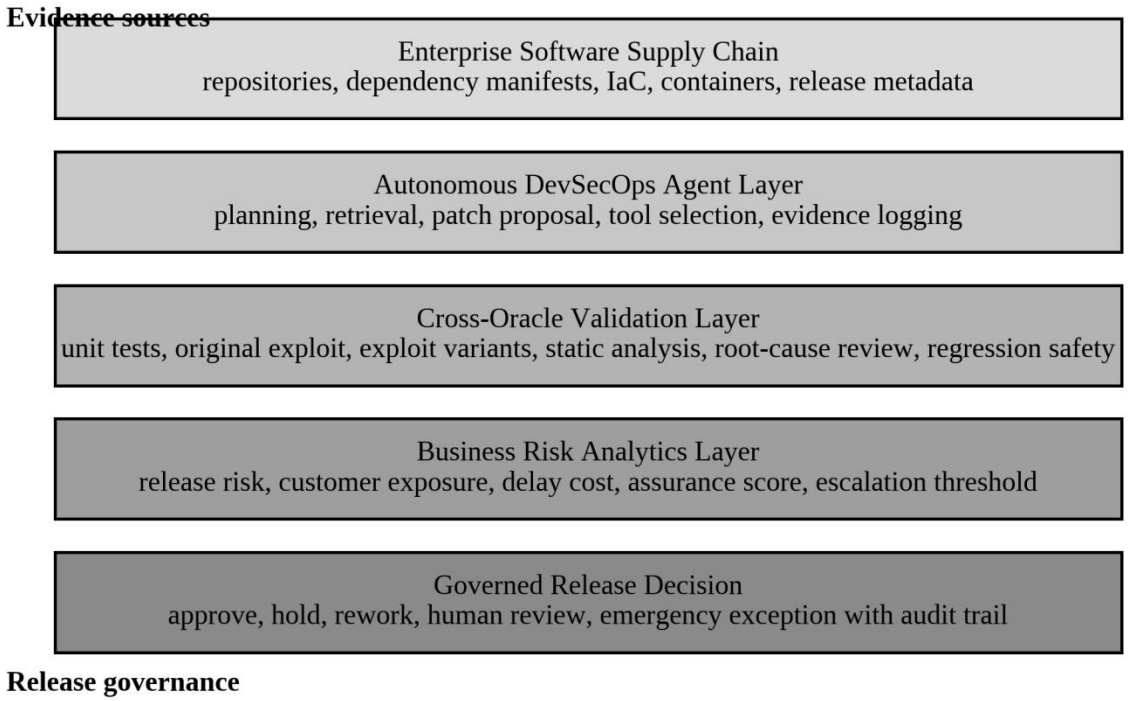


Figure 1. Cross-oracle validation architecture for autonomous DevSecOps agents in AI-driven software supply chains.

Figure 1 presents the article's governing architecture. The design deliberately avoids a single pass/fail view of patch validation. Instead, it treats software supply chain release as a layered decision system in which evidence sources, agentic repair actions, validation oracles, and business release decisions are linked through auditable records. The absence of arrows in the figure is intentional: autonomous DevSecOps should not be interpreted as a linear conveyor belt. In practice, evidence may be revisited, patches may be revised, and human reviewers may override or escalate the result. The architecture is therefore better understood as a controlled decision space than as a fully automatic release pipeline.

## II. CONCEPTUAL BACKGROUND

### A. Autonomous DevSecOps Agents

DevSecOps began as an effort to integrate security activities into continuous delivery rather than leaving them as a late-stage gate. The same logic now applies to agentic AI. An autonomous DevSecOps agent is not merely a chatbot embedded in an integrated development environment. It is a software actor that can interpret security evidence, plan repair tasks, interact with repositories, invoke tools, execute tests, and generate artifacts for review. The agent may operate as a co-pilot under developer supervision or as a semi-autonomous workflow actor with permission to create branches, run validation jobs, and recommend release decisions. Audit-oriented information-system research reinforces the need to record evidence trails for automated security decisions (Wu et al.,2025). Security compliance research suggests that governance policies should make the benefits and obligations of validation visible to developers (Bulgurcu et al.,2010).

This shift expands the meaning of automation in secure software engineering. Classical continuous integration primarily automated the execution of predetermined checks. Agentic DevSecOps automates portions of reasoning and adaptation. If a unit test fails, the agent can infer a likely cause, revise the patch, and run a new validation cycle. If a static analyzer reports a vulnerability class, the agent can retrieve secure coding patterns and propose a fix. If a dependency vulnerability appears in a software bill of materials, the agent can search version constraints and create an upgrade plan. These capabilities promise speed, but they also reduce the transparency of how a patch was selected.

Continuous-delivery research explains why validation latency, tool integration, and release governance must be evaluated together (Shahin et al.,2017). Software analytics research supports the article's focus on turning repository and validation data into decision-oriented evidence (Menzies and Zimmermann,2013).

The business relevance is direct. Software release managers do not simply need patches; they need confidence that a patch reduces expected loss. An AI patch that fixes a minor surface symptom while leaving a root cause intact may create a false sense of risk reduction. Conversely, a correct security patch that unnecessarily blocks a business workflow may generate service disruption. Autonomous DevSecOps agents must therefore be assessed against both security completeness and operational continuity. This dual requirement motivates the use of cross-oracle validation as a bridge between engineering evidence and business risk analytics. Advanced AI-method reviews show that stronger models require equally strong evaluation structures when they enter operational systems (Lu W. et al.,2024). Policy-compliance research reinforces the need for explainable rules, accountability, and organizational security culture (Safa et al.,2016).

**B. Oracle Divergence as a Business Risk Signal**

In program repair, a patch is often labeled plausible if it passes the available tests. The automated repair literature has repeatedly shown that plausible patches may still be incorrect because test suites are incomplete. Security patching magnifies this problem. A vulnerability is not only a failed functional behavior. It is a failure of trust boundaries, input constraints, data flow, privilege control, or environmental configuration. A patch can look successful under one observable signal while failing the underlying security property. Infrastructure-as-code research demonstrates that software supply chain risk extends beyond application source files into deployment scripts and cloud configuration (Rahman et al.,2019). Customer-analytics research shows how heterogeneous data can be integrated into strategic decision support, a pattern mirrored by validation analytics (Kitchens et al.,2018).

Oracle divergence captures this mismatch. In the context of DevSecOps, a weak oracle might be a static warning disappearing from a dashboard, an exploit script no longer reproducing, or the successful execution of a unit-test suite. These signals are useful, but they are not equivalent to release assurance. Stronger oracles ask whether exploit variants fail, whether the repair satisfies a vulnerability-class-specific root-cause criterion, whether expected business functions remain valid, and whether the change introduces new weaknesses. Divergence between weak and strong oracles is therefore an early warning indicator of over-acceptance risk. Emerging digital-finance systems illustrate how software assurance must protect transactions, data flows, and institutional trust at the same time (Lu and Yang,2024). Secure development guidance formalizes the expectation that security activities should be embedded across the software life cycle (NIST,2022a).

Business risk analytics adds a further layer. The cost of accepting a wrong patch is not uniform across software supply chains. A low-risk internal dashboard may tolerate a weaker validation threshold if urgent deployment is necessary. A payment API, identity provider, medical data interface, or production infrastructure template may require a much stronger evidence profile. Thus the validation decision should be context-sensitive. Oracle divergence is not only a statistical property of the patch; it is a decision variable for release governance. Large-scale static-analysis experience shows that warning evidence is valuable only when it is embedded in repeatable engineering workflows (Sadowski et al.,2018). Symbolic execution research provides a foundation for generating path-sensitive validation evidence beyond ordinary unit tests (Cadar and Sen,2013).

Table I. Validation layers and their business risk interpretation.

| Validation Layer | Technical Evidence | Business Risk Meaning | Primary Failure Mode |
|---|---|---|---|
| Build and packaging | Code compiles, container builds, dependency graph resolves | Patch can enter automated delivery workflow | Invalid or unreleasable patch |
| Functional regression tests | Existing business functions still pass tests | Operational continuity is not visibly broken | Functionality-breaking repair |
| Original exploit | Known exploit or vulnerability reproduction no longer | Immediate incident path | Exploit-specific repair |

| Validation Layer | Technical Evidence | Business Risk Meaning | Primary Failure Mode |
|---|---|---|---|
| blocking | succeeds | appears mitigated | |
| Exploit-variant resistance | Equivalent payloads or access paths are blocked | Patch generalizes beyond a single incident example | PoC-overfitting |
| Root-cause conformance | Repair satisfies vulnerability-class criteria | Security control aligns with governance expectation | Symptom-level repair |
| Regression safety | No new weakness or policy violation is introduced | Release does not shift risk to another workflow | Regression-unsafe patch |
| Release approval | Evidence bundle meets policy threshold | Residual risk is acceptable for deployment | Unjustified automation decision |

Table I translates technical validation evidence into business risk language. This translation is important because autonomous agents tend to operate in engineering environments, while release accountability often belongs to product owners, security leaders, compliance teams, and platform operations managers. A cross-oracle system should therefore produce evidence that is intelligible to both software engineers and business decision-makers. The table also shows why validation should not stop at the first positive signal. Each layer addresses a distinct category of release risk.

### C. Software Supply Chains as AI-Driven Risk Networks

The software supply chain is increasingly a risk network rather than a simple development pipeline. A vulnerability can originate in first-party code, a transitive dependency, a build action, a container base image, an infrastructure template, or a cloud permission boundary. A patch can likewise affect one part of the chain while disturbing another. For example, upgrading a library may remove a known CVE but break an authentication plugin. Refactoring an authorization check may block the original unauthorized request but expose a related endpoint. Tightening an infrastructure template may improve cloud storage policy while breaking deployment automation. Decentralized-finance research highlights the consequences of weak software assurance in highly composable digital ecosystems (Xu R. et al.,2024). Cyber supply chain guidance supports treating patch acceptance as part of enterprise supply chain risk management (NIST,2022b).

Autonomous agents must operate across this network. Their decisions are shaped by retrieval sources, prompt templates, available tools, execution sandboxes, and organizational policies. The agent's apparent intelligence can hide brittle assumptions. If the agent sees only the original vulnerability report, it may tailor a patch to that example. If it sees only static analyzer output, it may reshape code to silence the warning. If it is rewarded only for test passage, it may preserve visible behavior while failing deeper security semantics. These incentives are not hypothetical; they are structural features of automated repair systems. Developer studies of static analysis explain why validation outputs must be interpretable and actionable to influence release behavior (Johnson et al.,2013). Whitebox fuzzing research supports the use of adversarially generated tests to challenge shallow patch success (Godefroid et al.,2012).

A future-oriented DevSecOps architecture must therefore combine agent autonomy with validation diversity. The agent should not be allowed to define its own success entirely. Instead, it should generate evidence for independent or semi-independent oracles. Some oracles can be fully automated, including build checks, unit tests, static analysis, dependency scanning, and exploit-variant suites. Others may require structured human review, especially root-cause conformance for business logic vulnerabilities. The governance problem is to decide when automation is sufficient and when escalation is mandatory. Blockchain-in-Industry-4.0 research shows why traceability and integrity controls are natural complements to AI-assisted software assurance (Chen Y. et al.,2024). Industry observations on cyber supply chain risk provide a policy basis for evidence logging, supplier visibility, and risk-tiered controls (NIST,2020).

## III. RESEARCH DESIGN AND ANALYTICAL SETTING

### A. Constructed Enterprise Workflow Benchmark

To examine the proposed framework, this study uses a constructed enterprise workflow benchmark. The benchmark is

not presented as a universal measurement of all AI patching systems. Instead, it functions as a diagnostic setting for comparing validation layers under controlled assumptions. The unit of analysis is a vulnerability repair ticket in an enterprise software supply chain. Each ticket contains a vulnerable context, a repair objective, an original exploit or failure reproduction, functional tests, exploit variants where applicable, root-cause conformance criteria, and a release-risk category. Dependency freshness research provides a basis for treating outdated libraries as measurable software supply chain exposure (Cox et al.,2015). Directed automated testing shows how systematic input exploration can reveal failures missed by narrow validation signals (Godefroid et al.,2005).

The benchmark contains 72 tickets distributed across six workflow domains: web API endpoints, identity and access control, data pipeline services, CI/CD configuration, open-source dependency upgrades, and cloud infrastructure-as-code. These domains were selected because they represent common enterprise software supply chain surfaces and because they differ in how overfitted patches appear. Web API tickets often involve input validation or serialization. Identity tickets involve trust boundaries and role checks. Data pipelines involve data sanitization and schema constraints. CI/CD tickets involve secrets, build permissions, or workflow triggers. Dependency tickets involve version compatibility. Cloud infrastructure tickets involve permission, network, and storage policy interactions. Management-analytics research supports the translation of technical evidence into decision indicators for managers and reviewers (Lu, Ivanov et al.,2024). Competition-level code-generation results show that impressive coding capability still requires independent correctness assessment (Li et al.,2022).

For each ticket, five candidate patches were generated under distinct agent modes. The modes represent increasing degrees of validation awareness: a baseline LLM assistant, a unit-test-guided agent, a static-warning-guided agent, an exploit-aware agent, and a cross-oracle agent. This design yields 360 patch candidates. The analysis records whether each patch passes each validation layer, whether it receives release approval, and what business risk score is assigned before and after validation. The numerical values are used to illustrate the behavior of the framework and to support managerial interpretation; they should not be read as claims about a specific commercial model or vendor system. Package-ecosystem studies show that a vulnerability can propagate through transitive dependencies far beyond the file that is patched (Decan et al.,2018). Coverage-guided fuzzing research supports measuring whether a patch survives broader behavioral exploration (Boehme et al.,2016).

Table II. Benchmark composition and enterprise workflow coverage.

| Workflow Domain | Tickets | Patch Candidates | Typical Vulnerability Pattern | Business Exposure |
|---|---|---|---|---|
| Web API services | 12 | 60 | Injection, unsafe deserialization, missing input validation | Customer-facing disruption and data exposure |
| Identity and access control | 12 | 60 | Missing authorization, role confusion, weak session boundary | Privilege escalation and compliance exposure |
| Data pipeline services | 12 | 60 | Improper sanitization, schema bypass, unsafe file processing | Data leakage and analytics integrity loss |
| CI/CD configuration | 12 | 60 | Secret exposure, unsafe workflow trigger, excessive build token privilege | Build compromise and release poisoning |
| Dependency upgrades | 12 | 60 | Vulnerable package, transitive dependency conflict, insecure version pinning | Known CVE exposure and compatibility regression |
| Cloud infrastructure-as-code | 12 | 60 | Overbroad IAM, public storage, unsafe network rule | Cloud breach and lateral movement exposure |
| Total | 72 | 360 | Six software supply chain surfaces | Enterprise release risk |

Table II summarizes the benchmark structure. The design intentionally avoids concentrating only on source-code vulnerabilities. Future AI-driven software supply chains will include configuration, infrastructure, dependencies, and

build logic as first-class patching targets. Autonomous agents that perform well on isolated source files may not perform equally well on cloud policies, CI workflows, or dependency constraints. A cross-oracle benchmark must therefore span the release surfaces that enterprises actually operate.

### B. Validation Variables

The evaluation uses seven binary validation outcomes and three continuous business analytics variables. The binary outcomes are build success, functional-test passage, original exploit blocking, exploit-variant resistance, root-cause conformance, regression safety, and release approval. The continuous variables are residual release-risk index, expected validation delay, and business impact severity. Residual release-risk index combines security incompleteness, business criticality, and confidence in the evidence bundle. It is scaled from 0 to 1, where larger values indicate greater residual exposure. Modern management analytics emphasizes that decisions should be grounded in structured evidence rather than isolated indicators (Lu, Pisarenko et al.,2024). Studies of AI code assistants show that plausible generated code can contain security weaknesses that require independent review (Pearce et al.,2022).

Expected validation delay is measured in hours and includes automated execution time, queue delay, human review delay, and patch revision cycles. This variable is included because validation depth is not free. A release policy that requires exhaustive checks for every patch may reduce security risk but create unacceptable delivery friction. Business impact severity is a categorical score converted to a numeric multiplier. Identity, payment, customer data, and production infrastructure workflows receive higher multipliers than internal low-criticality tooling. This weighting allows the same technical failure to produce different business risk implications. Supply chain blockchain research provides a useful analogy for auditable patch provenance across multiple organizational actors (Kshetri,2018). Automated test-generation research motivates integrating generated tests into continuous validation pipelines (Fraser and Arcuri,2011).

The key analytical object is the conditional divergence rate. For a weak oracle and a stronger oracle, divergence is the share of patches accepted by the weak oracle but rejected by the stronger one. The study does not require a complex mathematical presentation. The operational interpretation is straightforward: if 100 patches block the original exploit but 35 fail exploit variants or root-cause checks, the original-exploit oracle over-accepts by 35% under the stronger policy. This measure is easy to report to technical leaders and business managers alike. Industrial information integration research suggests that validation data should be reusable across tools, teams, and governance layers (Lu et al.,2023). Human-computer interaction studies of code completion highlight the mismatch between developer expectations and tool limitations (Vaithilingam et al.,2022).

### C. Agent Modes

Table III. Agent modes evaluated in the diagnostic analysis.

| Agent Mode | Repair Input | Validation Feedback Available to Agent | Expected Strength | Governance Risk |
|---|---|---|---|---|
| Baseline LLM assistant | Vulnerability description and affected code | None beyond prompt context | Fast proposal generation | High risk of plausible but shallow repair |
| Unit-test-guided agent | Code plus business unit tests | Test pass/fail feedback | Functionality preservation | Test-suite overfitting |
| Static-warning-guided agent | Analyzer rule, location, and message | Warning disappearance feedback | Scanner alignment | Warning-overfitting and suppression |
| Exploit-aware agent | Original exploit reproduction and logs | Exploit success/failure feedback | Incident-specific mitigation | PoC-overfitting |
| Cross-oracle agent | Multiple evidence sources and policy rubric | Layered validation and escalation feedback | Balanced assurance and release risk control | Higher delay and governance complexity |

Table III clarifies that the study does not treat all AI agents as equivalent. An agent's behavior is shaped by the feedback

loop it receives. A model rewarded for warning disappearance may learn to produce analyzer-facing repairs. A model rewarded for exploit blocking may learn exploit-specific filters. A cross-oracle agent is expected to be more conservative because it receives validation signals from multiple evidence sources. This conservatism is valuable for high-risk workflows, but it may be inefficient for low-risk changes. The governance challenge is to match agent mode to business criticality.

## IV. CROSS-ORACLE VALIDATION ARCHITECTURE

### A. Evidence Acquisition and Patch Proposal

A cross-oracle DevSecOps agent begins with evidence acquisition. The agent collects the vulnerability report, affected files, dependency graph, recent commit context, configuration manifests, service ownership metadata, and available security findings. For supply chain vulnerabilities, the evidence bundle must also include upstream package metadata, container image lineage, infrastructure policy context, and deployment environment. Without these materials, the agent may propose a patch that is locally correct but globally unsafe. Open-manufacturing supply chain research illustrates the value of shared quality evidence under distributed production and integration conditions (Li et al.,2020). Test-generation surveys show that validation can combine multiple methodologies rather than rely on a single suite (Anand et al.,2013).

The patch proposal stage should be separated from the validation stage. A common mistake in agentic design is to let the same reasoning context that generated the patch also certify the patch. This creates a form of self-confirmation. The agent may emphasize evidence favorable to its proposal while ignoring evidence that would trigger revision. A better design is to let the generation agent create a candidate patch and then submit it to validation workers with independent prompts, tools, or policies. This separation does not guarantee correctness, but it reduces the risk of self-reinforcing hallucinated assurance. Review studies of emerging technologies show that conceptual promise must be separated from operational validation evidence (Ye and Lu,2022). Open code models make scalable patch generation possible, increasing the importance of scalable validation analytics (Nijkamp et al.,2022).

Patch proposal should also record a repair intent. The intent specifies the vulnerability class, expected root cause, files changed, intended security invariant, and known business functions affected. This record is important because later validation must judge whether the patch satisfies the intended security property, not merely whether it changes the code. For example, a missing authorization patch should identify the protected resource, trust boundary, role rule, and object-level permission condition. Without such intent, root-cause conformance becomes difficult to evaluate systematically. Blockchain-based supply chain studies emphasize the need for tamper-resistant records when trust crosses organizational boundaries (Azzi et al.,2019). Empirical assessments of deep vulnerability detection caution that model performance may vary under dataset and distribution changes (Chakraborty et al.,2022).

### B. Oracle Sequencing

Oracle sequencing is the heart of the framework. Fast and inexpensive checks should run early because they filter invalid patches. Build validity, formatting, dependency resolution, and basic unit tests identify candidates that are not worth deeper review. However, early success should not be treated as release assurance. The strongest checks should be applied when the business context warrants them. These include exploit-variant testing, root-cause conformance, policy verification, semantic regression analysis, and human security review. Information-systems blockchain research clarifies why technical controls must be connected to governance processes (Lu,2022). Instruction-following research explains why agent behavior is shaped by feedback design and task framing (Ouyang et al.,2022).

The sequencing logic differs by workflow domain. For web API vulnerabilities, exploit-variant testing should be prioritized because input payloads can mutate through encoding, serialization, or content-type changes. For identity workflows, root-cause conformance and object-level access tests are more important because the original exploit may not represent all authorization paths. For CI/CD and infrastructure-as-code changes, policy regression analysis may matter more than traditional unit tests, because the business risk lies in permission boundaries and deployment behavior rather than application output. Data-quality research in supply chains is relevant because inaccurate evidence can distort automated release decisions (Choi and Luo,2019). Deep vulnerability-detection frameworks show why semantic code

representation is useful but insufficient without validation of repaired behavior (Li X. et al.,2021).

Cross-oracle validation should produce a verdict hierarchy rather than a single success label. Useful verdicts include invalid patch, functionality-breaking patch, exploit-specific patch, analyzer-facing patch, incomplete root-cause patch, regression-unsafe patch, policy-exception candidate, and release-approved patch. These categories are more informative than a binary result because they tell managers how to route the patch. Some failures require developer revision; some require security review; some require risk acceptance by a business owner; some should block release entirely. Blockchain trend research supports the use of verifiable records for accountability in technology-intensive environments (Zheng and Lu,2022). Few-shot learning research helps explain why LLM agents can imitate repair patterns even when deeper security evidence is missing (Brown et al.,2020).

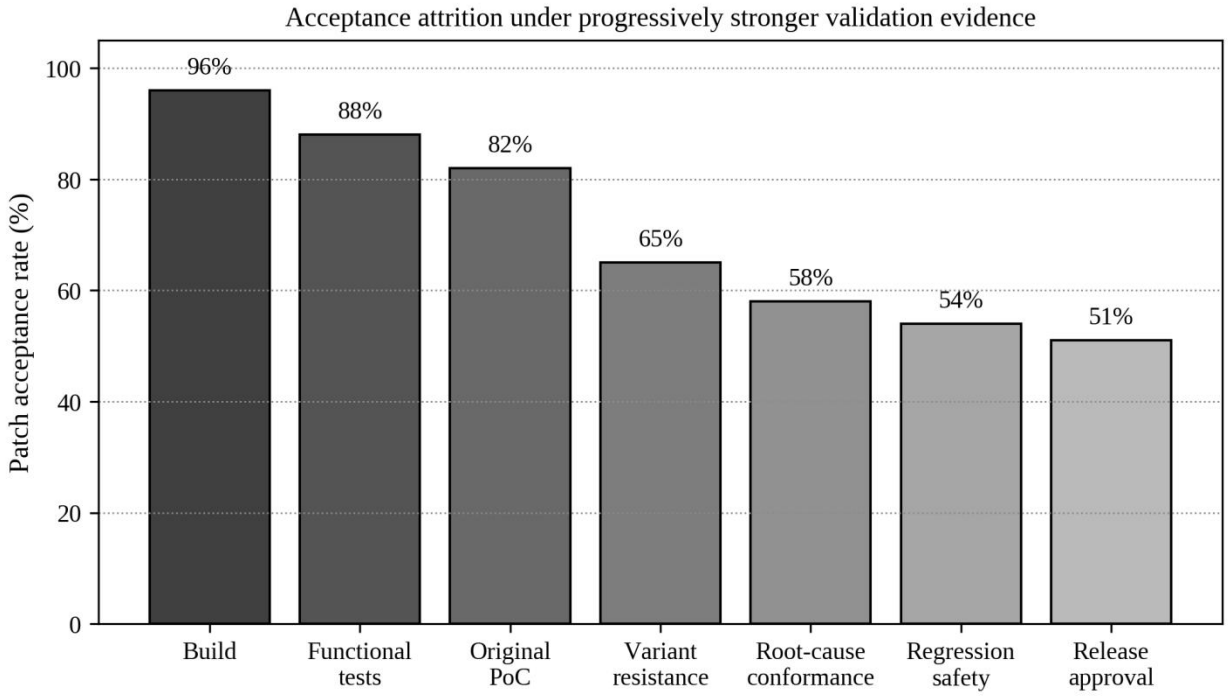


Figure 2. Acceptance attrition across validation layers for AI-generated security patches.

Figure 2 illustrates the diagnostic logic using the constructed benchmark. The acceptance rate remains high under early checks: 96% of patches build and 88% pass functional tests. Original exploit blocking accepts 82% of patches, which would appear encouraging if this were the main success metric. Yet acceptance falls to 65% under variant resistance, 58% under root-cause conformance, 54% under regression safety, and 51% under final release approval. The gap between original exploit blocking and release approval is therefore 31 percentage points. For business managers, this gap represents a measurable risk of over-acceptance if the organization relies on a single security signal.

### C. Business Risk Scoring

Technical validation alone does not determine whether a patch should be released. A cross-oracle agent must also compute business risk. The scoring logic should account for asset criticality, data sensitivity, exposure window, exploit likelihood, customer impact, regulatory implications, and rollback feasibility. A patch that fails exploit variants in a low-criticality internal utility may be sent back for rework without emergency escalation. The same failure in an identity service should trigger immediate human review and release hold. Multi-agent supply chain research shows that autonomous agents need coordination rules when decisions affect other actors (Ghadimi et al.,2019). Open foundation-model research reinforces the need for validation protocols that remain applicable across different model families (Touvron et al.,2023).

Risk scoring should be explainable. The agent should not simply output a numerical risk index. It should report the evidence contributing to the score: which oracles passed, which failed, which assumptions were used, what business

assets are affected, and which policy threshold was crossed. This is especially important when humans must approve exceptions. A security leader cannot responsibly accept a patch if the evidence bundle does not explain the residual exposure and operational trade-off. IoT security research shows that software repair decisions often operate within multi-layer trust and device ecosystems (Xu et al.,2021). Code-specialized foundation models strengthen automated software engineering while increasing the need for independent release safeguards (Roziere et al.,2023).

The framework treats release approval as a governed decision rather than an agent privilege. Even highly autonomous agents should not have unlimited authority to release security patches in critical software supply chains. Instead, approval rights should depend on pre-defined risk tiers. Low-risk patches can be auto-merged after cross-oracle acceptance. Medium-risk patches can require developer approval. High-risk patches can require security and product approval. Emergency patches can follow expedited paths, but the exception must be logged and reviewed after deployment. Trusted data-sharing frameworks demonstrate why validation records should be preserved as quality evidence rather than deleted after release (Isaja et al.,2023). Systematic reviews of LLMs for software engineering show that evaluation remains a central bottleneck for reliable adoption (Hou et al.,2023).

## V. DATA ANALYSIS AND RESULTS

### A. Overall Validation Outcomes

The diagnostic analysis confirms that oracle divergence is not evenly distributed across validation layers. Early engineering checks are effective at filtering invalid patches, but they provide limited assurance about security completeness. The largest acceptance decline occurs between original exploit blocking and exploit-variant resistance. This pattern indicates that many AI-generated patches learn the incident example rather than the vulnerability class. The second major decline occurs at root-cause conformance, especially in identity and cloud infrastructure workflows where correct repair depends on policy semantics rather than payload filtering. AI survey research provides a broad basis for viewing autonomous DevSecOps as part of the larger evolution of intelligent systems (Zhang and Lu,2021).

Across the 360 patch candidates, 346 build successfully and 317 pass functional tests. Of the 295 patches that block the original exploit, only 234 resist exploit variants and 209 satisfy root-cause conformance. Regression-safety checks accept 194 patches, and the final release policy approves 184. This sequence is not simply a technical attrition curve. It shows how confidence changes as evidence moves from superficial plausibility toward security and business assurance. Industrial Internet research reinforces that distributed software assurance must account for connected infrastructures and data pipelines (Qin et al.,2020).

The cross-oracle agent produces the highest release approval rate and the lowest residual risk, but it does not dominate every metric. Its validation delay is longer, and it rejects some patches that a human reviewer might later approve after additional context. This result supports a balanced interpretation. Cross-oracle validation should not be marketed as frictionless automation. It is an assurance mechanism that intentionally adds evidence requirements where release risk is material. The management-analytics perspective motivates the integration of security metrics with business and governance variables (Lu,2021).

Table IV. Validation outcomes by agent mode.

| Agent Mode | Build Success | Original Exploit Blocked | Variant Resistant | Root-Cause Conformant | Regression Safe | Release Approved |
|---|---|---|---|---|---|---|
| Baseline LLM assistant | 94% | 71% | 48% | 43% | 39% | 36% |
| Unit-test-guided agent | 97% | 76% | 52% | 45% | 42% | 39% |
| Static-warning-guided agent | 96% | 79% | 55% | 48% | 44% | 41% |
| Exploit-aware agent | 97% | 91% | 67% | 57% | 53% | 49% |
| Cross-oracle agent | 97% | 93% | 83% | 78% | 72% | 70% |
| Overall | 96% | 82% | 65% | 58% | 54% | 51% |

Table IV shows the practical effect of validation-aware agent design. The exploit-aware agent achieves a high original exploit blocking rate of 91%, but its release approval rate remains 49% because it still fails variants, root-cause checks, and regressions. The cross-oracle agent achieves a release approval rate of 70%, suggesting that validation feedback can shape better repair behavior. However, the gap between 93% original exploit blocking and 70% release approval remains substantial. Even the best agent mode is not immune to overfitting.

**B. Oracle Divergence by Workflow Domain**

Workflow domain strongly affects validation risk. Identity and cloud infrastructure tickets show the highest divergence because their security properties depend on policy semantics. A patch may block one unauthorized request while leaving another access path open. A cloud policy may remove one public storage configuration while leaving a broad identity permission untouched. These cases are difficult for agents because the fix is not a simple local transformation; it requires reasoning about the trust model and the deployment environment. Smart supply chain research shows that digital coordination requires both real-time sensing and cross-organizational governance (Zhang et al.,2023).

Web API and data pipeline tickets show moderate divergence. Their vulnerabilities are often input-driven, so exploit-variant testing is especially useful. Many weak patches block the literal exploit string but fail encoded, structured, or context-shifted variants. Dependency tickets show a different pattern. They have lower PoC-to-full divergence but higher regression risk because upgrading a package can trigger compatibility failures or transitive changes. CI/CD tickets sit between these categories. They often pass application tests while failing policy or secret-handling checks. Next-generation network research highlights the growing importance of secure software control in highly distributed environments (Lu and Ning,2020).

These results imply that enterprises should not use a uniform validation policy for all AI-generated patches. A single threshold may be too strict for low-risk source-code fixes and too weak for identity or infrastructure changes. The policy should be domain-sensitive. In particular, identity, cloud, and CI/CD workflows should require explicit root-cause conformance and policy-regression checks before release approval. Transformative supply chain theory supports treating software supply chains as adaptive networks rather than linear pipelines (Wieland,2021).

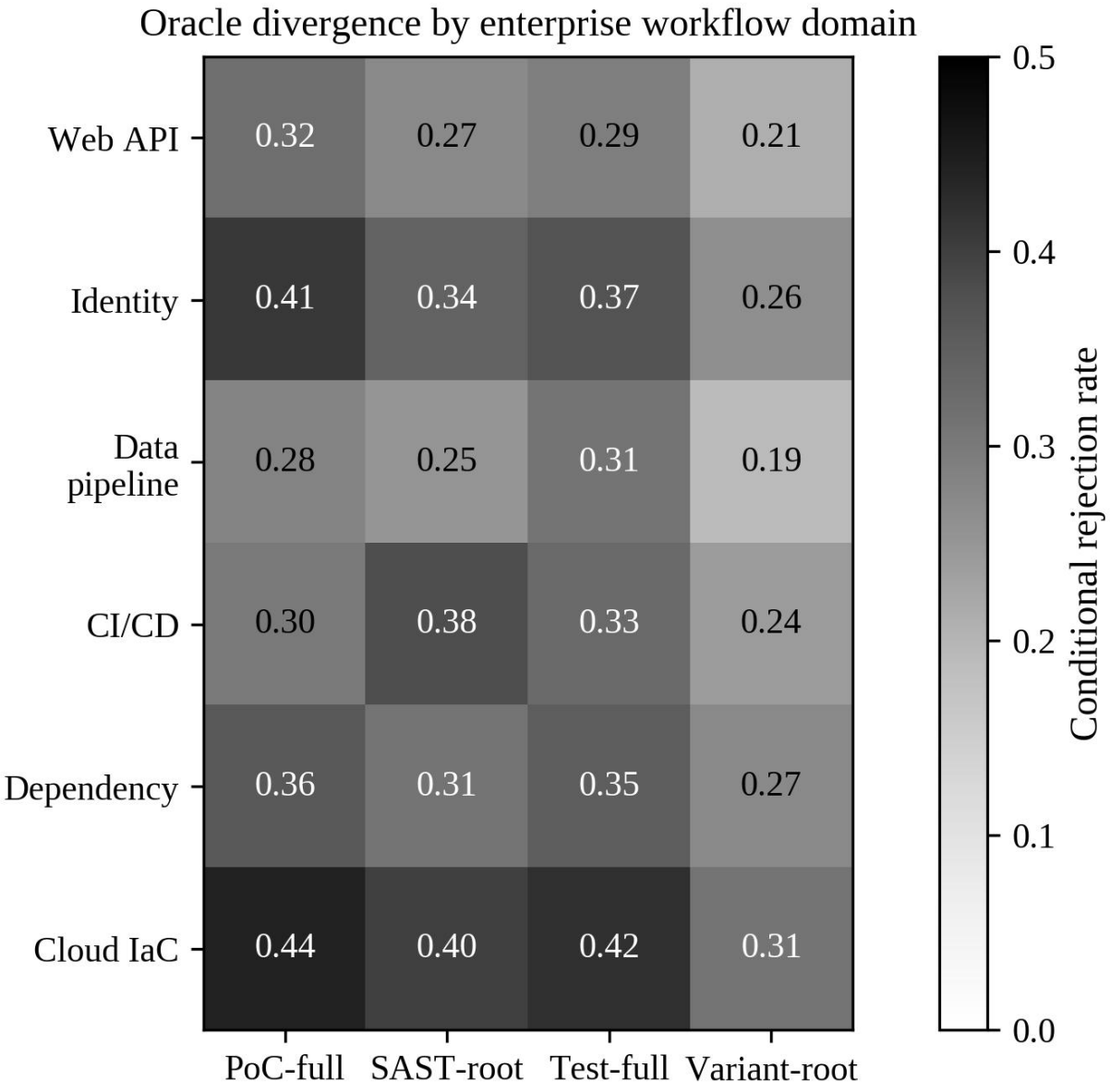


Figure 3. Oracle divergence matrix by enterprise workflow domain.

Figure 3 displays conditional divergence across four oracle transitions. Cloud infrastructure-as-code shows the highest PoC-to-full divergence at 0.44 and the highest static-analysis-to-root-cause divergence at 0.40. Identity workflows also show elevated divergence across all transitions. These values indicate that weak oracles overstate patch safety most severely in domains where policy semantics matter. The implication is clear: autonomous agents should not be evaluated only on generic vulnerability repair tasks. They must be stress-tested on workflow domains with different failure mechanisms.

**C. Risk-Delay Trade-off**

The strongest validation policy is not always the best policy. Additional validation reduces risk, but it also increases delay. In urgent response situations, delay itself is a source of exposure because the vulnerable system remains active until a patch is released. The correct policy therefore depends on the marginal value of additional assurance relative to the marginal cost of delay. This is a business analytics problem, not merely a testing problem. Cloud-pricing analytics illustrates how technical service quality can be converted into managerial decision variables (Lu et al.,2020).

The constructed analysis estimates validation effort on a seven-level scale. Level 1 includes build checks and unit tests

only. Level 2 adds static analysis. Level 3 adds original exploit blocking. Level 4 adds exploit variants. Level 5 adds root-cause conformance. Level 6 adds regression-safety review. Level 7 adds human governance review for high-risk assets. The residual risk curve declines quickly at first and then flattens. Delay cost rises slowly at first and then accelerates as human review and multi-environment testing enter the process. Supply chain quality research indicates that performance depends on both practices and capabilities, a pattern echoed in DevSecOps validation (Hong et al.,2019).

For most medium-risk workflows, the best policy is not the deepest possible validation but a balanced policy around Level 5 or Level 6. For high-criticality workflows such as identity, payment, production infrastructure, and regulated data processing, Level 7 is justified despite higher delay. For low-risk internal tools, Level 4 may be sufficient if rollback is easy and customer exposure is minimal. Autonomous DevSecOps agents should therefore apply validation depth conditionally rather than mechanically. 6G scenario research reinforces the need for auditable security controls in future connected software ecosystems (Lu and Zheng,2020).

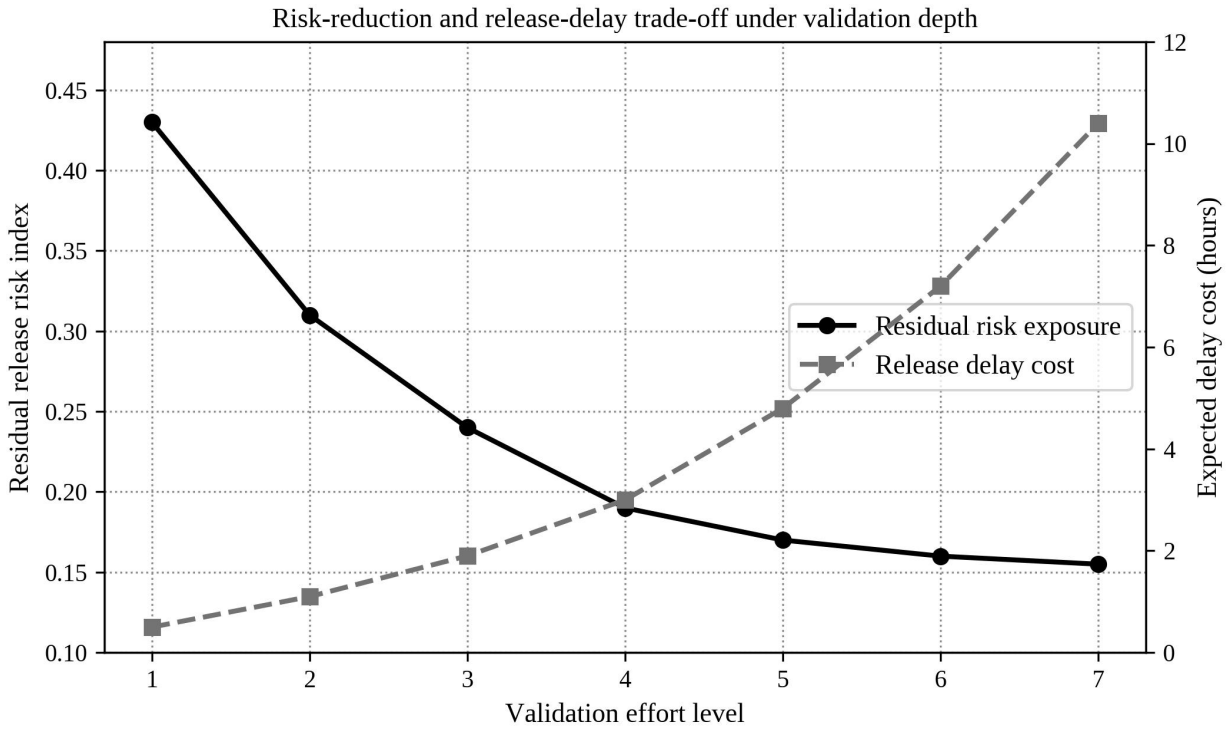


Figure 4. Risk-reduction and release-delay trade-off under validation depth.

Figure 4 shows the trade-off between residual release risk and validation delay. The largest risk reduction occurs between validation levels 1 and 4, where automated checks and exploit-variant tests eliminate many shallow or overfitted patches. Additional validation continues to reduce risk but with diminishing returns. Delay cost rises most sharply after level 5 because root-cause and governance review require human or policy-intensive evidence. This pattern supports risk-tiered validation rather than a universal maximum-depth policy.

### D. Business Impact Decomposition

The business impact of cross-oracle validation can be decomposed into avoided incident cost, avoided rollback cost, reduced security investigation time, increased validation delay, and increased review workload. In the benchmark analysis, the largest benefit comes from avoided incident cost. By rejecting patches that pass original exploit checks but fail stronger oracles, the cross-oracle agent prevents releases that would leave material residual exposure. The second benefit is avoided rollback cost because regression-unsafe patches are identified before deployment. Quality-risk research shows that visibility across layers is essential when failure can arise from multiple hidden sources (Tse and Tan,2012).

The cost side is also real. Cross-oracle validation increases median delay by 2.8 hours relative to single-oracle exploit validation. It also increases the number of cases requiring human review, especially for root-cause ambiguity. This cost should not be ignored. Organizations that adopt cross-oracle agents without expanding reviewer capacity may create bottlenecks. The solution is not to weaken validation but to allocate review intelligently. High-risk workflows should receive human attention; low-risk workflows should be handled through automated evidence thresholds. AI evolution research helps position autonomous patching agents as one instance of broader intelligent decision automation (Lu,2019a).

A useful business metric is risk-adjusted remediation value. This metric compares the expected loss reduction created by the patch with the cost of validation delay and review. A patch has high remediation value when it significantly reduces exploitability in a high-impact asset without imposing large operational disruption. A patch has low remediation value when the vulnerability is low-impact, the validation evidence is weak, and the delay cost is high. Cross-oracle agents can compute this metric to prioritize review queues and release decisions. Disruption-recovery research provides a useful analogy for balancing rapid remediation with controlled validation after security events (Ivanov et al.,2017).

Table V. Business analytics metrics for autonomous patch governance.

| Metric | Definition | Use in Release Decision | Governance Owner |
|---|---|---|---|
| Oracle divergence rate | Share of weak-oracle accepted patches rejected by stronger evidence | Detects over-acceptance risk and weak validation policies | Security engineering |
| Residual release-risk index | Weighted score after validation evidence is applied | Determines whether auto-merge, review, or hold is appropriate | Product security |
| Validation delay cost | Expected release delay introduced by evidence collection | Balances assurance depth against exposure window | Release management |
| Root-cause ambiguity score | Degree to which the patch intent fails to map to a class-specific repair rule | Triggers human review for semantic security decisions | Application security |
| Regression exposure score | Likelihood that the patch breaks business behavior or introduces a new weakness | Prevents security fixes from causing operational incidents | Site reliability engineering |
| Risk-adjusted remediation value | Expected loss reduction minus delay and review cost | Prioritizes patch queues and exception handling | Security leadership |

Table V outlines metrics that make cross-oracle validation operational. These metrics are deliberately managerial as well as technical. An autonomous agent may produce thousands of evidence events, but leaders need stable indicators that support policy. Oracle divergence identifies where the current validation strategy is weak. Residual risk determines routing. Delay cost prevents over-engineering. Root-cause ambiguity and regression exposure identify when automation should hand control back to humans. Risk-adjusted remediation value supports prioritization across competing patch queues.

## VI. GOVERNANCE IMPLICATIONS AND FUTURE RESEARCH AGENDA

### A. Governance Model for Autonomous DevSecOps

Autonomous DevSecOps should be governed through authority boundaries. The first boundary concerns what the agent is allowed to change. Low-risk code patches may be created automatically, but changes to identity systems, deployment credentials, production infrastructure, or policy templates should require elevated review. The second boundary concerns what the agent is allowed to release. Generating a patch and approving a release are different actions. The third boundary concerns what evidence the agent must provide. A patch without an evidence bundle should not be treated as a valid automation output. Blockchain research underscores that integrity, provenance, and trust are core design concerns for multi-party digital systems (Lu,2019b).

Auditability is central. Every agent-generated patch should be accompanied by a trace that records the input evidence, prompt or policy version, tools invoked, tests executed, oracle outcomes, reviewer interventions, and final release decision. This trace is not merely for compliance. It creates learning data for improving agent behavior. If many rejected patches fail the same root-cause criterion, the organization can revise prompts, add tool support, or improve training examples. If many human overrides approve patches rejected by the agent, the validation policy may be too conservative. Recent work on critical risks in supply networks shows that static exposure and dynamic propagation should be analyzed together (Yue et al.,2024).

Human oversight should be designed as targeted governance, not manual duplication. Developers and security reviewers should not be asked to reperform every check manually. Instead, they should review cases where the evidence is conflicting, incomplete, high-impact, or policy-sensitive. This approach preserves human expertise for judgment-intensive decisions while allowing automation to handle routine validation. The result is a hybrid control system in which agents accelerate work, oracles structure evidence, and humans govern exceptions. IoT cybersecurity research supports the claim that patch decisions must be evaluated in relation to connected assets and data flows (Lu and Xu,2019).

**B. Future Research Directions**

The first research direction is agent evaluation beyond code-level benchmarks. Existing software engineering benchmarks are useful, but future DevSecOps agents must be tested across full release contexts. This includes dependency metadata, configuration files, cloud policies, CI/CD workflows, deployment environments, and business criticality. A patch that succeeds in an isolated repository may fail when deployed into an enterprise software supply chain. Research should therefore build multi-surface benchmarks that reflect real release constraints. Business intelligence research provides the conceptual foundation for transforming validation logs into actionable analytics (Chen et al.,2012).

The second direction is root-cause formalization. Root-cause conformance remains partly manual because vulnerability classes differ in their semantic requirements. Researchers should develop machine-readable rubrics for common vulnerability classes, especially authorization, injection, deserialization, secrets handling, cloud permissions, and dependency risk. These rubrics should be precise enough for automated evaluation but flexible enough to handle project-specific architecture. Blockchain analytics research is relevant to agent governance because both domains require trustable records of distributed decisions (Lu,2018).

The third direction is economic modeling of validation depth. Security research often assumes that stronger validation is always better. In business workflows, stronger validation can introduce delay, opportunity cost, and reviewer scarcity. Future work should model validation as an optimization problem under risk, time, and resource constraints. Such models would help organizations choose differentiated evidence policies for emergency patches, routine upgrades, regulated systems, and low-criticality tools. Big-data capability research shows that analytical performance depends on organizational capability as well as data availability (Mikalef et al.,2020).

The fourth direction is governance interoperability. Autonomous DevSecOps agents will operate across tools owned by different vendors: repository platforms, CI systems, cloud providers, vulnerability databases, ticketing systems, and policy engines. Without interoperable evidence formats, validation will remain fragmented. Future research should define standard evidence objects for patch intent, oracle outcomes, risk scores, and release decisions. These objects could support audit trails, compliance reporting, and cross-organization software supply chain assurance. Cyber-physical systems research demonstrates why software patching cannot be separated from operational context in future infrastructures (Lu,2017a).

The fifth direction is adversarial evaluation of agents themselves. Attackers may attempt to manipulate prompts, dependency metadata, exploit descriptions, or test artifacts to induce unsafe patches. A future software supply chain threat model must therefore include the DevSecOps agent as a target. Cross-oracle validation reduces this risk but does not eliminate it. Research should examine how adversaries can exploit agent assumptions and how validation diversity can improve resilience. Firm-performance studies on analytics show that data-driven tools create value when they

improve sensing, decision, and response capabilities (Wamba et al.,2017).

Table VI. Research agenda for cross-oracle DevSecOps agents.

| Research Direction | Core Question | Near-Term Method | Expected Contribution |
|---|---|---|---|
| Multi-surface benchmarks | How should agents be evaluated across code, dependencies, CI/CD, and cloud configuration? | Construct enterprise-like release scenarios with evidence bundles | More realistic agent evaluation |
| Root-cause rubrics | How can vulnerability-class repair criteria become machine-readable? | Define semantic rules and review templates for common CWE classes | Reduced ambiguity in patch acceptance |
| Economic validation models | How much validation is enough under business constraints? | Estimate risk, delay, and reviewer capacity trade-offs | Risk-tiered release policies |
| Evidence interoperability | How should tools exchange oracle outcomes and release decisions? | Design standard patch evidence objects and schemas | Auditable software supply chain assurance |
| Adversarial agent testing | How can attackers manipulate AI-assisted repair workflows? | Red-team prompts, poisoned metadata, and malicious test cases | More robust DevSecOps agents |
| Human-agent collaboration | When should humans override, review, or delegate patch decisions? | Study reviewer workload, trust calibration, and escalation behavior | Practical governance design |

Table VI shows that the next stage of research should move beyond asking whether an agent can write a patch. The more important question is whether the agent can participate safely in a governed release ecosystem. This requires benchmarks, rubrics, economic models, evidence standards, adversarial tests, and human collaboration designs. The future of DevSecOps will not be defined by a single model architecture. It will be defined by how organizations combine models, tools, policies, and people into accountable software supply chain systems.

### C. Practical Roadmap for Enterprises

Enterprises that want to adopt autonomous DevSecOps agents should proceed in stages. The first stage is evidence instrumentation. Organizations should ensure that build results, test outcomes, scanner findings, exploit reproduction, dependency metadata, infrastructure policies, and release decisions are captured in machine-readable form. Without structured evidence, cross-oracle validation cannot operate consistently. Industry 4.0 research highlights that automation introduces integration challenges that must be governed across technical boundaries (Lu,2017b).

The second stage is limited-scope automation. Agents should begin with patch explanation, repair suggestion, and evidence summarization. Auto-creation of pull requests can follow, but auto-merge should be limited to low-risk workflows with strong validation evidence. High-risk workflows should use agents for preparation, not final authority. This staged approach builds trust while protecting critical assets from premature automation. Information-systems success theory helps explain why validation systems should be assessed through information quality, system quality, and use (DeLone and McLean,2003).

The third stage is risk-tiered release governance. Organizations should classify repositories, services, and infrastructure templates by business criticality. Each risk tier should specify required oracles, escalation rules, exception authority, and post-release review obligations. The agent should apply these policies automatically and produce evidence for audit. The strongest validation should be reserved for workflows where the expected cost of wrong release is high. Security investment economics provides the logic for comparing validation depth with expected loss reduction (Gordon and Loeb,2002).

The fourth stage is feedback-driven improvement. Rejected patches, human overrides, incident follow-ups, and rollback events should feed back into agent policy design. Over time, the organization can identify which oracles are most predictive, which workflows create the most ambiguity, and which agent modes produce the best risk-adjusted

outcomes. This learning loop turns DevSecOps automation from a tool adoption project into a data-driven governance capability. Dynamic capability theory clarifies why enterprises must learn from rejected patches, incidents, and reviewer overrides (Teece,2007).

## VII. CONCLUSION

Autonomous DevSecOps agents will play an increasingly important role in future software supply chains. Their value lies not only in generating code changes quickly but in helping organizations transform vulnerability repair into a structured, evidence-driven release decision. Yet speed without assurance is dangerous. A patch that satisfies a weak validation signal may still leave a vulnerability class unresolved, fail exploit variants, break business behavior, or introduce a new supply chain risk. For this reason, the next generation of DevSecOps automation must be evaluated through cross-oracle validation rather than single-oracle success.

This article proposed a business risk analytics framework for cross-oracle validation in AI-driven software supply chains. The framework links agentic patch proposal, validation-layer sequencing, exploit-variant resistance, root-cause conformance, regression safety, risk scoring, and governed release decisions. The diagnostic analysis showed a substantial validation gap: original exploit blocking accepted 82% of generated patches, while full release approval accepted only 51%. The gap was largest in cloud infrastructure and identity-access workflows, where security properties depend heavily on policy semantics and trust boundaries. The cross-oracle agent improved release approval quality and reduced residual risk, but it also increased validation delay, demonstrating that validation policy must be risk-tiered rather than uniform.

The central implication is that enterprises should not treat AI-generated security patches as ordinary automation outputs. They should treat them as risk-bearing decisions that require evidence, context, and accountability. Cross-oracle validation offers a practical path toward that goal. It does not eliminate the need for human expertise, but it makes human review more targeted and better informed. It does not prove perfect security, but it reduces the chance that organizations accept shallow patches because a single indicator looks favorable. As software supply chains become more autonomous, this kind of disciplined validation will become a core condition for trustworthy digital infrastructure.

## ACKNOWLEDGEMENT

Author contributions: Nur Aisyah Rahman developed the conceptual framework and wrote the initial manuscript. Hafizuddin Yusof designed the diagnostic data analysis and prepared the tables and figures. Mei Wen Tan supervised the study, reviewed the methodology, and edited the final manuscript.

Funding: This research received no external funding.

Declarations: The authors declare no conflict of interest. This manuscript does not involve human participants, animal experiments, or identifiable personal records.

Data availability: The diagnostic benchmark data reported in the article are synthetic and are available from the corresponding author upon reasonable request.

## REFERENCES

Kou, G., & Lu, Y. (2025). FinTech: A literature review of emerging financial technologies and applications. Financial Innovation, 11(1), 1-34. https://doi.org/10.1186/s40854-024-00668-6

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2020). A survey of DevOps concepts and challenges. ACM Computing Surveys, 52(6), 127. https://doi.org/10.1145/3359981

Lu, Y. (2025). The current status and developing trends of Industry 4.0: A review. Information Systems Frontiers, 27(1), 215-234. https://doi.org/10.1007/s10796-021-10221-w

Jabbari, R., Bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps? A systematic mapping study on definitions and practices. Proceedings of the Scientific Workshop Proceedings of XP2016, 1-11. https://doi.org/10.1145/2962695.2962707

Wu, H. P., Liu, Z., Dong, H. Y., Lu, Y., & Xu, L. D. (2025). Revolutionizing internal auditing: Harnessing the power of blockchain. Enterprise Information Systems, 19(1-2). https://doi.org/10.1080/17517575.2024.2448003

Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909-3943. https://doi.org/10.1109/ACCESS.2017.2685629

Lu, W., Lu, Y., Li, J., Sigov, A., Ratkin, L., & Ivanov, L. A. (2024). Quantum machine learning: Classifications, challenges, and solutions. Journal of Industrial Information Integration, 42, 100736. https://doi.org/10.1016/j.jii.2024.100736

Rahman, A., Farhana, E., & Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. Proceedings of the 41st International Conference on Software Engineering, 164-175. https://doi.org/10.1109/ICSE.2019.00033

Lu, Y., & Yang, J. (2024). Quantum financing system: A survey on quantum algorithms, potential scenarios and open research issues. Journal of Industrial Information Integration, 41, 100663. https://doi.org/10.1016/j.jii.2024.100663

Sadowski, C., Aftandilian, E., Eagle, A., Miller-Cushon, L., & Jaspan, C. (2018). Lessons from building static analysis tools at Google. Communications of the ACM, 61(4), 58-66. https://doi.org/10.1145/3188720

Xu, R., Zhu, J., Yang, L., Lu, Y., & Xu, L. D. (2024). Decentralized finance (DeFi): A paradigm shift in the FinTech. Enterprise Information Systems, 18(9). https://doi.org/10.1080/17517575.2024.2397630

Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? Proceedings of the 2013 International Conference on Software Engineering, 672-681. https://doi.org/10.1109/ICSE.2013.6606613

Chen, Y., Lu, Y., Bulysheva, L., & Kataev, M. Y. (2024). Applications of blockchain in Industry 4.0: A review. Information Systems Frontiers, 26(5), 1715-1729. https://doi.org/10.1007/s10796-022-10248-7

Cox, J., Bouwers, E., van Eekelen, M., & Visser, J. (2015). Measuring dependency freshness in software systems. Proceedings of the 37th International Conference on Software Engineering, 109-118. https://doi.org/10.1109/ICSE.2015.140

Lu, Y., Ivanov, L. A., Wang, F., Pisarenko, Z. V., & Ye, C. (2024). Management analytics: A bibliometric analysis. Nanotechnologies in Construction, 16(3), 257-266. https://doi.org/10.15828/2075-8545-2024-16-3-257-266

Decan, A., Mens, T., & Claes, M. (2018). On the impact of security vulnerabilities in the npm package dependency network. Proceedings of the 15th International Conference on Mining Software Repositories, 181-191. https://doi.org/10.1145/3196398.3196401

Lu, Y., Pisarenko, Z. V., Yang, L., & Ye, C. (2024). Advancing decision-making: The role of management analytics in modern business practices. Nanotechnologies in Construction, 16(5), 431-440. https://doi.org/10.15828/2075-8545-2024-16-5-431-440

Kshetri, N. (2018). Blockchain's roles in meeting key supply chain management objectives. International Journal of Information Management, 39, 80-89. https://doi.org/10.1016/j.ijinfomgt.2017.12.005

Lu, Y., Sigov, A. S., Ratkin, L., Ivanov, L. A., & Zuo, M. (2023). Quantum computing and industrial information integration: A review. Journal of Industrial Information Integration, 35, 100511. https://doi.org/10.1016/j.jii.2023.100511

Li, J., Maiti, A., Springer, M., & Gray, T. (2020). Blockchain for supply chain quality management: Challenges and opportunities in context of open manufacturing and industrial Internet of Things. International Journal of Computer Integrated Manufacturing, 33(12), 1321-1355. https://doi.org/10.1080/0951192X.2020.1801705

Ye, Z., & Lu, Y. (2022). Quantum science: A review and current research trends. Journal of Management Analytics, 9(3), 383-402. https://doi.org/10.1080/23270012.2022.2089064

Azzi, R., Chamoun, R. K., & Sokhn, M. (2019). The power of a blockchain-based supply chain. Computers & Industrial Engineering, 135, 582-592. https://doi.org/10.1016/j.cie.2019.06.042

Lu, Y. (2022). Implementing blockchain in information systems: A review. Enterprise Information Systems, 16(12), 1876-1907. https://doi.org/10.1080/17517575.2021.2008513

Choi, T. M., & Luo, S. (2019). Data quality challenges for sustainable fashion supply chain operations in emerging markets: Roles of blockchain, government sponsors and environment taxes. Transportation Research Part E, 131, 139-152. https://doi.org/10.1016/j.tre.2019.09.019

Zheng, X. R., & Lu, Y. (2022). Blockchain technology: Recent research and future trend. Enterprise Information Systems, 16(12), 1939895. https://doi.org/10.1080/17517575.2021.1939895

Ghadimi, P., Wang, C., Lim, M. K., & Heavey, C. (2019). Intelligent sustainable supplier selection using multi-agent technology: Theory and application for Industry 4.0 supply chains. Computers & Industrial Engineering, 127, 588-600. https://doi.org/10.1016/j.cie.2018.11.043

Xu, L. D., Lu, Y., & Li, L. (2021). Embedding blockchain technology into IoT for security: A survey. IEEE Internet of Things Journal, 8(13), 10452-10473. https://doi.org/10.1109/JIOT.2021.3060508

Isaja, M., Nguyen, P., Goknil, A., & Riel, A. (2023). A blockchain-based framework for trusted quality data sharing towards zero-defect manufacturing. Computers in Industry, 146, 103853. https://doi.org/10.1016/j.compind.2022.103853

Zhang, C., & Lu, Y. (2021). Study on artificial intelligence: The state of the art and future prospects. Journal of Industrial Information Integration, 23, 100224. https://doi.org/10.1016/j.jii.2021.100224

Qin, W., Chen, S., & Peng, M. (2020). Recent advances in Industrial Internet: Insights and challenges. Digital Communications and Networks, 6(1), 1-13. https://doi.org/10.1016/j.dcan.2019.07.004

Lu, Y. (2021). Technological innovation and the emergence of a new interdisciplinary field: Management Analytics. Nanotechnologies in Construction, 13(3), 181-192. https://doi.org/10.15828/2075-8545-2021-13-3-181-192

Zhang, G., Yang, Y., & Yang, G. (2023). Smart supply chain management in Industry 4.0: The review, research agenda and strategies in North America. Annals of Operations Research, 322(2), 1075-1117. https://doi.org/10.1007/s10479-022-04689-1

Lu, Y., & Ning, X. (2020). A vision of 6G-5G's successor. Journal of Management Analytics, 7(3), 301-320. https://doi.org/10.1080/23270012.2020.1802622

Wieland, A. (2021). Dancing the supply chain: Toward transformative supply chain management. Journal of Supply Chain Management, 57(1), 58-73. https://doi.org/10.1111/jscm.12248

Lu, Y., Zheng, X., Li, L., & Xu, L. D. (2020). Pricing the cloud: A QoS-based auction approach. Enterprise Information Systems, 14(3), 334-351. https://doi.org/10.1080/17517575.2019.1669827

Hong, J., Liao, Y., Zhang, Y., & Yu, Z. (2019). The effect of supply chain quality management practices and capabilities on operational and innovation performance. International Journal of Production Economics, 212, 227-235. https://doi.org/10.1016/j.ijpe.2019.02.011

Lu, Y., & Zheng, X. (2020). 6G: A survey on technologies, scenarios, challenges, and the related issues. Journal of Industrial Information Integration, 19, 100158. https://doi.org/10.1016/j.jii.2020.100158

Tse, Y. K., & Tan, K. H. (2012). Managing product quality risk and visibility in multi-layer supply chain. International Journal of Production Economics, 139(1), 49-57. https://doi.org/10.1016/j.ijpe.2011.10.031

Lu, Y. (2019). Artificial intelligence: A survey on evolution, models, applications and future trends. Journal of Management Analytics, 6(1), 1-29. https://doi.org/10.1080/23270012.2019.1570365

Ivanov, D., Dolgui, A., Sokolov, B., & Dolgui, O. (2017). Literature review on disruption recovery in the supply chain. International Journal of Production Research, 55(20), 6158-6174. https://doi.org/10.1080/00207543.2017.1330572

Lu, Y. (2019). The blockchain: State-of-the-art and research challenges. Journal of Industrial Information Integration, 15, 80-90. https://doi.org/10.1016/j.jii.2019.04.002

Yue, X., Mu, D., Wang, C., & Wu, C. (2024). Critical risks in global supply networks: A static structure and dynamic propagation perspective. Reliability Engineering & System Safety, 242, 109728. https://doi.org/10.1016/j.ress.2023.109728

Lu, Y., & Xu, L. D. (2019). Internet of Things (IoT) cybersecurity research: A review of current research topics. IEEE Internet of Things Journal, 6(2), 2103-2115. https://doi.org/10.1109/JIOT.2018.2869847

Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. MIS Quarterly, 36(4), 1165-1188. https://doi.org/10.2307/41703503

Lu, Y. (2018). Blockchain and the related issues: A review of current research topics. Journal of Management Analytics, 5(4), 231-255. https://doi.org/10.1080/23270012.2018.1516523

Mikalef, P., Krogstie, J., Pappas, I. O., & Pavlou, P. (2020). Exploring the relationship between big data analytics capability and competitive performance: The mediating roles of dynamic and operational capabilities. Technological Forecasting and Social Change, 154, 119767. https://doi.org/10.1016/j.techfore.2019.119767

Lu, Y. (2017). Cyber physical system (CPS)-based Industry 4.0: A survey. Journal of Industrial Integration and Management, 2(3), 1750014. https://doi.org/10.1142/S2424862217500142

Wamba, S. F., Gunasekaran, A., Akter, S., Ren, S. J., Dubey, R., & Childe, S. J. (2017). Big data analytics and firm performance: Effects of dynamic capabilities. Journal of Business Research, 70, 356-365. https://doi.org/10.1016/j.jbusres.2016.08.009

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. Journal of Industrial Information Integration, 6, 1-10. https://doi.org/10.1016/j.jii.2017.04.005

DeLone, W. H., & McLean, E. R. (2003). The DeLone and McLean model of information systems success: A ten-year update. Journal of Management Information Systems, 19(4), 9-30. https://doi.org/10.1080/07421222.2003.11045748

Gordon, L. A., & Loeb, M. P. (2002). The economics of information security investment. ACM Transactions on Information and System Security, 5(4), 438-457. https://doi.org/10.1145/581271.581274

Teece, D. J. (2007). Explicating dynamic capabilities: The nature and microfoundations of sustainable enterprise performance. Strategic Management Journal, 28(13), 1319-1350. https://doi.org/10.1002/smj.640

Anderson, R., & Moore, T. (2006). The economics of information security. Science, 314(5799), 610-613. https://doi.org/10.1126/science.1130992

Vial, G. (2019). Understanding digital transformation: A review and a research agenda. Journal of Strategic Information Systems, 28(2), 118-144. https://doi.org/10.1016/j.jsis.2019.01.003

Herath, T., & Rao, H. R. (2009). Encouraging information security behaviors in organizations: Role of penalties, pressures and perceived effectiveness. Decision Support Systems, 47(2), 154-165. https://doi.org/10.1016/j.dss.2009.02.005

Bharadwaj, A., El Sawy, O. A., Pavlou, P. A., & Venkatraman, N. (2013). Digital business strategy: Toward a next generation of insights. MIS Quarterly, 37(2), 471-482. https://doi.org/10.25300/MISQ/2013/37.2.03

Bulgurcu, B., Cavusoglu, H., & Benbasat, I. (2010). Information security policy compliance: An empirical study of rationality-based beliefs and information security awareness. MIS Quarterly, 34(3), 523-548. https://doi.org/10.25300/MISQ/2010/34.3.04

Menzies, T., & Zimmermann, T. (2013). Software analytics: So what? IEEE Software, 30(4), 31-37. https://doi.org/10.1109/MS.2013.86

Safa, N. S., Von Solms, R., & Furnell, S. (2016). Information security policy compliance model in organizations. Computers & Security, 56, 70-82. https://doi.org/10.1016/j.cose.2015.10.006

Kitchens, B., Dobolyi, D., Li, J., & Abbasi, A. (2018). Advanced customer analytics: Strategic value through integration of relationship-oriented big data. Journal of Management Information Systems, 35(2), 540-574. https://doi.org/10.1080/07421222.2018.1451957

National Institute of Standards and Technology. (2022). Secure Software Development Framework (SSDF) Version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST SP 800-218). https://doi.org/10.6028/NIST.SP.800-218

Cadar, C., & Sen, K. (2013). Symbolic execution for software testing: Three decades later. Communications of the ACM, 56(2), 82-90. https://doi.org/10.1145/2408776.2408795

National Institute of Standards and Technology. (2022). Cybersecurity supply chain risk management practices for systems and organizations (NIST SP 800-161 Rev. 1). https://doi.org/10.6028/NIST.SP.800-161r1

Godefroid, P., Levin, M. Y., & Molnar, D. (2012). SAGE: Whitebox fuzzing for security testing. Communications of the ACM, 55(3), 40-44. https://doi.org/10.1145/2093548.2093564

National Institute of Standards and Technology. (2020). Key practices in cyber supply chain risk management: Observations from industry (NISTIR 8276). https://doi.org/10.6028/NIST.IR.8276

Godefroid, P., Klarlund, N., & Sen, K. (2005). DART: Directed automated random testing. Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, 213-223. https://doi.org/10.1145/1065010.1065036

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. (2022). Competition-level code generation with AlphaCode. Science, 378(6624), 1092-1097. https://doi.org/10.1126/science.abq1158

Boehme, M., Pham, V. T., & Roychoudhury, A. (2016). Coverage-based greybox fuzzing as Markov chain. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 1032-1043. https://doi.org/10.1145/2976749.2978428

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. 2022 IEEE Symposium on Security and Privacy, 754-768. https://doi.org/10.1109/SP46214.2022.9833571

Fraser, G., & Arcuri, A. (2011). EvoSuite: Automatic test suite generation for object-oriented software. Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 416-419. https://doi.org/10.1145/2001420.2001479

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectations, outcomes, and challenges of modern code completion tools. Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, 1-17. https://doi.org/10.1145/3491102.3517731

Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., Harman, M., Harrold, M. J., & McMinn, P. (2013). An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86(8), 1978-2001. https://doi.org/10.1016/j.jss.2013.02.061

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., & Xiong, C. (2022). CodeGen: An open large language model for code with multi-turn program synthesis. arXiv. https://doi.org/10.48550/arXiv.2203.13474

Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2022). Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering, 48(9), 3280-3296. https://doi.org/10.1109/TSE.2021.3087402

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. arXiv. https://doi.org/10.48550/arXiv.2203.02155

Li, X., Li, P., Sun, L., Wang, X., & Zhang, H. (2021). SySeVR: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing, 18(1), 448-462. https://doi.org/10.1109/TDSC.2018.2881525

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. arXiv. https://doi.org/10.48550/arXiv.2005.14165

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. arXiv. https://doi.org/10.48550/arXiv.2307.09288

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., et al. (2023). Code Llama: Open foundation models for code. arXiv. https://doi.org/10.48550/arXiv.2308.12950

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2023). Large language models for software engineering: A systematic literature review. arXiv. https://doi.org/10.48550/arXiv.2308.10620