International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 2, 193-204, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I2P127

Original Article

LLM-Driven DevSecOps for Secure Software Engineering: Integrating Static Analysis, Retrieval-Augmented Generation, and Automated Vulnerability Remediation

Dr. S. Ganesh Kumar[1], Dr. Robert .P[2], Dr. Suganya A[3], Dr. Rama .P[4]
[1]Professor, CS, Data Science & Business Systems, SRM Institute of Science and Technology, Chennai, Tamilnadu, India.
[2]Assistant Professor, CS, CSE, SRM Institute of Science and Technology, Chennai, Tamilnadu, India.
[3]Assistant Professor, AI & ML, Computer Vision, SRM Institute of Science and Technology, Chennai, Tamilnadu, India.
[4]Assistant Professor, AI & ML, Computing Technologies, SRM Institute of Science and Technology, Chennai, Tamilnadu, India.

Received On: 24/03/2026 ***Revised On: 18/04/2026*** ***Accepted On: 25/04/2026*** ***Published On: 06/05/2026***

Abstract *- The rapid adoption of large language models in software engineering has transformed how developers design, generate, review, test, and maintain code. However, the same automation that accelerates development also introduces security risks when generated patches, dependency upgrades, or code explanations are accepted without systematic verification. This paper proposes an LLM-driven DevSecOps framework that integrates static application security testing, retrieval-augmented generation, secure knowledge grounding, automated vulnerability remediation, human approval workflows, and continuous post-remediation validation. The central thesis is that LLMs should not be treated as autonomous security authorities, but as governed remediation agents operating within evidence-based, policy-constrained, and auditable software delivery pipelines. The framework aligns secure software development with NIST SSDF practices by embedding security activities across planning, implementation, verification, release, and monitoring stages [1]. It also extends prior work on AI-driven software lifecycle governance by positioning defect prediction, automated testing, vulnerability detection, and remediation as mutually reinforcing components rather than isolated quality engineering tasks [2]. The proposed architecture uses static analysis findings as structured triggers, retrieval-augmented generation as an evidence-grounded reasoning layer, and controlled patch generation as a remediation mechanism. The paper further presents a methodological design for evaluating the framework using vulnerability detection precision, remediation correctness, regression safety, developer review burden, mean time to remediate, and auditability. The contribution is a reference architecture for secure AI-assisted engineering that reduces manual triage effort while preserving accountability, traceability, and security assurance.*

Keywords **-** *Devsecops, Large Language Models, Static Analysis, Retrieval-Augmented Generation, Vulnerability Remediation, Secure Software Engineering, Software Supply Chain, AI Governance, Automated Program Repair, Secure Coding.*

1. Introduction

Modern software engineering has entered a phase in which the boundary between development support and autonomous code transformation is becoming increasingly blurred. Large language models are now used to generate application logic, explain unfamiliar code, write tests, refactor legacy components, summarize pull requests, and suggest vulnerability fixes.

This adoption is not merely an efficiency trend; it is reshaping the security model of software delivery because security decisions are increasingly influenced by probabilistic systems that may generate plausible but incorrect reasoning. Retrieval-augmented generation has become an important mechanism for reducing hallucination by grounding model outputs in external knowledge sources, policy documents, vulnerability taxonomies, secure coding standards, and repository-specific context [3].

DevSecOps emerged to address the historical separation between development velocity and security assurance. In classical pipelines, static analysis, dependency scanning, secret detection, infrastructure-as-code validation, container scanning, and dynamic testing often generate large volumes of findings that exceed the review capacity of engineering teams. When alerts are noisy, developers may postpone remediation or suppress warnings without fully understanding exploitability.

Banking and regulated software environments demonstrate that intelligent continuous integration and delivery requires risk scoring, deployment evaluation, and governance-driven release control rather than simple pass-fail automation [4]. In security-sensitive domains, the operational value of DevSecOps is therefore determined not only by tool coverage, but by the ability to translate findings into prioritized, explainable, and verifiable engineering actions.

The central challenge is that LLM-based remediation can amplify both the strengths and weaknesses of DevSecOps. An LLM may quickly explain a SQL injection finding, identify vulnerable code paths, propose a parameterized query, and generate unit tests. At the same time, it may introduce incomplete validation, unsafe defaults, broken business logic, or dependency changes that pass narrow tests but fail in production-like scenarios. Secure microservices engineering in regulated environments shows that architectural controls, deployment boundaries, encryption, compliance requirements, and runtime isolation must be treated as part of the remediation context rather than as afterthoughts [5]. This paper therefore argues that LLM remediation must be constrained by static evidence, retrieved security knowledge, repository-specific architecture, policy checks, and human approval

The proposed framework is built around four principles. First, static analysis remains the primary evidence source for vulnerability discovery and localization. Second, RAG provides controlled access to trusted knowledge, including CWE descriptions, OWASP categories, internal secure coding standards, architectural decision records, dependency policies, and prior approved fixes. Third, automated remediation must be patch-oriented, test-bound, and reversible. Fourth, every LLM action must be logged with prompt inputs, retrieved evidence, model output, patch diff, validation results, reviewer decision, and deployment outcome. This model draws inspiration from code-language representation research, where models such as CodeBERT demonstrate that program understanding benefits from joint natural-language and programming-language representations [6].

The paper contributes a Q1-style conceptual and technical architecture for LLM-driven DevSecOps. It does not claim that LLMs can replace secure engineering teams. Instead, it positions LLMs as governed assistants for vulnerability explanation, patch proposal, regression-test generation, and remediation documentation. Prior research on code metrics and feature extraction for vulnerability detection supports the premise that vulnerability remediation can be improved when code-level signals are captured in structured form and used for downstream reasoning [7]. In this paper, those structured signals become the bridge between static analysis and LLM remediation.

2. Background and Related Work

Secure software engineering has historically relied on a layered combination of preventive design, secure coding standards, manual review, automated scanning, penetration testing, and operational monitoring. Static application security testing remains a foundational layer because it analyzes source code, bytecode, or intermediate representations before deployment. However, static tools are constrained by false positives, limited semantic understanding, framework-specific blind spots, and difficulty explaining business impact. Explainable AI research in regulated fraud detection demonstrates that automated decisions become more useful when they include auditable decision pathways and interpretable evidence trails [8]. Similar explainability is needed when DevSecOps systems recommend code changes.

Cloud-native and edge environments further complicate secure remediation because vulnerabilities may arise from interactions among services, APIs, identity providers, message brokers, containers, and external dependencies. Multi-tenant edge cloud research highlights the importance of efficient task placement, resource isolation, and adaptive infrastructure decisions under operational constraints [9]. In DevSecOps, remediation must similarly account for runtime constraints such as latency, secrets management, network policies, and deployment topology. A fix that is locally correct at the code level may still be insecure if it violates service authorization rules, weakens observability, or bypasses policy controls.

LLM security has become a distinct concern because model-integrated applications introduce new attack surfaces. The OWASP GenAI Security Project identifies risks across LLM application design, including prompt injection, sensitive information disclosure, insecure output handling, excessive agency, and vector or embedding weaknesses [10]. These risks are directly relevant to LLM-driven DevSecOps because the remediation agent itself may be attacked through repository content, malicious comments, poisoned documentation, crafted issue descriptions, or adversarial pull request text. As a result, DevSecOps agents require defensive prompt design, retrieval filtering, output validation, and strict execution boundaries.

Testing research remains central to the safety of automated remediation. Comparative work on Java enterprise testing frameworks shows that reliability assurance depends on selecting appropriate test types, coverage goals, and automation layers [11]. In LLM-driven remediation, generated patches should never be accepted solely because they compile. They must pass unit tests, regression tests, security tests, dependency checks, integration tests, and when needed, contract tests. The test suite becomes the enforcement layer that transforms LLM output from a suggestion into a controlled software engineering artifact.

Enterprise AI platforms also show that scalable and explainable AI requires governance mechanisms that extend beyond model accuracy. Studies of machine learning frameworks in CRM platforms emphasize secure scaling, explainability, and operational lifecycle management [12]. This perspective applies directly to LLM DevSecOps: the value of the model depends not only on its ability to generate a patch, but also on whether the organization can control the model's inputs, outputs, permissions, evaluation criteria, and deployment behavior. The system must be designed for governance from the beginning.

Research on code representation has improved the ability of machine learning systems to reason about software structure. GraphCodeBERT extends sequence-based

program representation by incorporating data-flow information, which is particularly relevant for vulnerability detection because many vulnerabilities depend on how untrusted data flows through program variables, sanitizers, sinks, and authorization checks [13]. Static analysis tools already construct control-flow and data-flow graphs. An LLM-driven system can use these artifacts as structured context, allowing the model to reason about vulnerable paths without relying only on raw source code.

Software defect prediction research provides another foundation for intelligent DevSecOps. Comparative analysis of machine learning models for defect prediction demonstrates that algorithm choice, feature selection, and evaluation design influence predictive reliability [14]. Vulnerability remediation has similar constraints: a model must not only identify a potential weakness, but also distinguish exploitable risks from theoretical warnings, prioritize high-impact issues, and recommend changes that reduce risk without causing regressions. This requires a pipeline that combines static findings, historical defect patterns, repository context, and validation outcomes.

Feature modeling and product-line integrity are also relevant because enterprise systems often contain configurable behavior across multiple tenants, products, or customer segments. Formal approaches to feature model integrity show that refactoring must preserve constraints across related configurations [15]. In DevSecOps, automated remediation must avoid breaking feature-specific logic when patching shared libraries or common services. A vulnerability fix in an authentication module may affect multiple applications, versions, and deployment configurations. Therefore, remediation should be dependency-aware and configuration-aware.

Recent benchmarks such as SWE-bench evaluate whether language models can resolve real-world GitHub issues by generating patches against existing repositories [16]. Although such benchmarks demonstrate progress, secure remediation requires a stricter standard than issue closure. A patch may satisfy a narrow functional test while leaving a vulnerability partially exploitable. Security patches often require negative tests, threat modeling, abuse-case validation, and regression checks against bypass variants. Therefore, LLM-driven DevSecOps must evaluate not only whether the patch solves the visible issue, but whether it reduces attack surface under adversarial conditions.

3. Problem Statement and Research Motivation

The research problem addressed in this paper is the absence of a governed architecture for integrating LLM-based remediation into DevSecOps pipelines without weakening security assurance. Existing DevSecOps tools detect vulnerabilities, but they often leave developers with fragmented alerts, limited remediation context, and high triage overhead. LLMs can explain and patch vulnerabilities, but they may hallucinate APIs, misunderstand architecture, generate insecure alternatives, or introduce subtle regressions. Security communication research in compliant fax and encrypted data transmission contexts illustrates that automated systems must preserve confidentiality, integrity, and operational auditability when handling sensitive workflows [17]. Similar requirements apply when LLMs inspect source code, logs, secrets, or vulnerability findings.

The first motivation is alert-to-action conversion. Static analysis tools identify tainted data flow, injection risks, insecure cryptographic usage, deserialization flaws, path traversal, broken access control, and unsafe dependency usage. However, many findings remain unresolved because developers lack time, context, or confidence. Machine learning anomaly detection in insurance systems demonstrates how domain-specific models can identify risk patterns that would be difficult to manage manually at scale [18]. DevSecOps needs the same transition from alert generation to intelligent action, but with stronger safeguards because remediation modifies production code.

The second motivation is knowledge fragmentation. Secure remediation requires information from multiple sources: source code, build files, dependency manifests, test results, architecture diagrams, secure coding standards, vulnerability databases, cloud policies, and prior incident records. LLMs trained on general code do not automatically know which internal libraries are approved, which patterns are prohibited, or which compliance obligations apply. Earlier work on the expanding role of ML models in software development anticipated that machine learning would increasingly influence development workflows, but the governance burden becomes larger when models produce executable code [19].

The third motivation is pipeline accountability. Automated remediation cannot be a black-box activity because security patches must be reviewed, justified, and auditable. Observability frameworks for cloud-based data pipelines show that monitoring, defect prediction, and operational visibility are essential for maintaining reliability across distributed systems [20]. LLM-driven remediation requires a comparable observability model: every generated patch should be traceable to the triggering finding, retrieved knowledge, model prompt, validation evidence, reviewer decision, and deployment result. Without this traceability, organizations cannot reliably assess whether AI-assisted remediation improves security or merely increases change volume.

The fourth motivation is RAG-specific security. RAG allows an LLM to use external knowledge, but vector stores and embedding pipelines can introduce new vulnerabilities. OWASP identifies vector and embedding weaknesses as risks in systems that use RAG, including manipulated retrieval, injected content, and exposure of sensitive information [21]. In DevSecOps, this risk is especially serious because repository documents may include comments, issue descriptions, or markdown files controlled by untrusted contributors. If a malicious document instructs the agent to ignore security policy or exfiltrate code, the

agent must reject that retrieved content and follow higher-priority governance rules.

The fifth motivation is adaptive infrastructure. Cloud systems increasingly use AI-augmented service fabrics to allocate resources and respond to operational conditions [22]. Secure remediation pipelines similarly need adaptive behavior. A low-risk documentation vulnerability may require only explanation and developer review, while a high-risk authentication flaw may require mandatory security approval, expanded tests, threat modeling, and staged deployment. The pipeline must classify vulnerability context and choose the appropriate remediation path.

Finally, the sixth motivation is modernization. Many enterprises still operate legacy monoliths alongside microservices, API gateways, cloud functions, and event-driven services. Fault-aware monolith decomposition research demonstrates that migration must manage gateway optimization, service boundaries, and reliability risks [23]. LLM-driven remediation must be compatible with such hybrid architectures. A patch suggested for a legacy module may need to respect transaction boundaries, shared database constraints, and downstream service contracts. Therefore, remediation should be architecture-aware, not merely syntax-aware.

4. Proposed LLM-Driven DevSecOps Architecture

The proposed architecture, called SecureRAG-Remediate, consists of seven layers: source ingestion, static analysis orchestration, vulnerability normalization, secure retrieval, LLM reasoning, controlled patch generation, and validation-governance. The design follows a governed agentic pattern in which the LLM is allowed to reason and propose changes but not independently merge or deploy security-sensitive code. Governed agentic AI research for enterprise platforms emphasizes data grounding, trust controls, decision intelligence, and lifecycle reliability as core architectural requirements [24]. SecureRAG-Remediate applies these same principles to software security engineering.

The source ingestion layer connects to repositories, branch metadata, dependency manifests, infrastructure-as-code files, container definitions, CI logs, test reports, and issue trackers. It computes a repository security profile that includes language, framework, dependency ecosystem, authentication patterns, data access mechanisms, and deployment environment. A systematic review of LLMs and code security notes that LLMs may both assist and harm secure coding because they can detect or fix vulnerabilities while also introducing insecure code [25]. The ingestion layer therefore treats all generated output as untrusted until validated.

The static analysis orchestration layer runs multiple analyzers based on repository type. For example, Java services may use taint analysis, dependency checking, secret scanning, and configuration scanning; Python services may use package vulnerability checks, unsafe deserialization detection, and static type-aware analysis; JavaScript applications may include DOM-based XSS checks, dependency audit, and API misuse detection. Hybrid deep learning models for software fault prediction show that combining CNN, LSTM, and dense layers can capture complementary signals from software artifacts [26]. In the proposed framework, static tools and learned models similarly provide complementary evidence.

The vulnerability normalization layer maps tool-specific findings to a common schema. The schema includes finding identifier, CWE category, severity, confidence, affected file, vulnerable line range, data-flow trace, sink type, exploit preconditions, affected dependency, reachable endpoint, business service, and recommended remediation pattern. Federated learning work for fraud detection highlights the value of privacy-preserving architecture and operational governance when multiple institutions or data owners are involved [27]. In secure engineering, vulnerability normalization allows teams to share remediation intelligence without exposing unnecessary proprietary code.

The secure retrieval layer is the core of the RAG design. It retrieves from four controlled repositories: public security knowledge, internal policy knowledge, repository-specific knowledge, and historical remediation knowledge. Public knowledge includes CWE descriptions, OWASP categories, NIST guidance, secure coding rules, and language-specific framework recommendations. Internal knowledge includes approved cryptographic libraries, authentication standards, logging rules, privacy requirements, and deployment constraints. Repository knowledge includes architecture diagrams, API contracts, test structure, and service ownership. Historical knowledge includes previously approved fixes and rejected remediation patterns. OCR accuracy research in healthcare prescription processing illustrates how domain-specific accuracy improves when models operate over structured, task-relevant inputs rather than generic text [28]. Secure retrieval follows the same principle by grounding remediation in curated and relevant evidence.

The LLM reasoning layer receives a structured prompt containing the normalized finding, minimized vulnerable code context, retrieved evidence, repository constraints, and required output format. It must produce an explanation, risk assessment, patch strategy, test plan, and proposed diff. To prevent excessive autonomy, the model is prohibited from changing unrelated files, disabling tests, weakening authentication, removing logging, adding unapproved dependencies, or suppressing scanner rules. Clean-core AI integration research in SAP systems emphasizes that AI enhancements should preserve architectural integrity rather than introduce uncontrolled customization [29]. This principle is translated into code-level guardrails for remediation.

The controlled patch generation layer generates candidate patches in isolated branches. The system may

produce multiple candidate patches, but each must include a rationale and validation plan. Backend AI-agent replacement discussions are relevant because enterprise automation must distinguish between agentic convenience and production accountability [30]. In this architecture, the LLM is not a replacement for developers; it is a patch proposal engine operating under approval and validation controls.

The validation-governance layer executes compilation, unit tests, regression tests, security tests, static rescans, dependency audits, and policy checks. It also performs semantic checks such as ensuring that the vulnerable sink is no longer reachable from untrusted input. Sparse matrix factorization research in scalable cloud machine learning suggests that efficient representation of high-dimensional relationships can improve performance in large systems [31]. Similarly, validation-governance uses compact representations of findings, files, dependencies, and tests to prioritize which validation jobs are necessary for a given patch.

5. Methodology and Evaluation Design

The proposed evaluation methodology is designed to measure security effectiveness, engineering usefulness, and governance quality. A secure remediation framework cannot be evaluated only by counting generated patches. It must be evaluated by whether vulnerabilities are correctly fixed, whether false positives are reduced, whether regressions are avoided, and whether developers trust the recommendations. AI-enhanced API reliability testing in digital banking shows that accuracy, resilience, and transaction integrity are critical when testing financial software workflows [32]. SecureRAG-Remediate adopts similarly strict evaluation criteria.

The experimental design uses three repository categories. The first category consists of benchmark repositories with known vulnerabilities, such as injection, path traversal, insecure deserialization, hardcoded secrets, broken access control, and vulnerable dependencies. The second category consists of enterprise-like synthetic microservices with realistic authentication, persistence, messaging, and deployment configurations. The third category consists of historical internal findings where the ground truth includes the human-approved remediation. End-to-end AI-based systems engineering research supports the use of lifecycle governance, predictive quality assurance, automation economics, and cybersecurity intelligence as combined evaluation dimensions [33].

The baseline systems include static analysis without LLM support, generic LLM prompting without retrieval, RAG without strict policy filtering, and the full SecureRAG-Remediate architecture. This comparison isolates the value added by retrieval grounding, policy constraints, and validation gates. Predictive analytics and Redis-backed caching research in pharmacy inventory management shows that performance and responsiveness depend on architectural choices such as caching, data freshness, and workload-specific optimization [34]. Similarly, RAG systems must be evaluated for retrieval latency, knowledge freshness, and context relevance.

The primary security metrics include true positive remediation rate, false remediation rate, residual vulnerability rate, patch correctness, exploit-path elimination, and scanner recurrence rate. Patch correctness is measured by whether the patch removes the vulnerability without changing intended behavior. Residual vulnerability rate captures cases where the generated patch appears to address the finding but leaves a bypass path. Research on neural network architecture optimization through evolutionary algorithms shows that model performance depends heavily on objective formulation and search constraints [35]. In this work, objective formulation means balancing vulnerability removal, minimal code change, test success, and maintainability.

The engineering productivity metrics include mean time to triage, mean time to remediate, developer review time, number of manual edits required after LLM patch generation, and pull request cycle time. Decision intelligence frameworks for Salesforce enterprise platforms demonstrate that predictive and prescriptive systems are valuable when they turn complex data into operational decisions [36]. The proposed framework applies this idea to DevSecOps by transforming scanner findings into prioritized engineering actions.

The quality assurance metrics include unit-test pass rate, regression-test pass rate, security-test pass rate, mutation-test sensitivity, code coverage change, and static re-scan cleanliness. Adaptive ensemble approaches in software fault detection emphasize balancing accuracy and robustness rather than optimizing one metric in isolation [37]. Secure remediation requires the same balance: a patch that removes one vulnerability but introduces unstable behavior is not acceptable. The architecture-awareness metrics include dependency impact, service impact, API contract preservation, configuration preservation, and feature-flag compatibility. Graph-based modeling of service dependencies helps predict failure propagation in distributed systems [38]. SecureRAG-Remediate uses service dependency graphs to determine whether a patch requires additional downstream tests or staged rollout.

The human governance metrics include reviewer acceptance rate, explanation usefulness, audit completeness, rollback frequency, and policy violation count. Probabilistic reasoning in multi-agent reinforcement learning is relevant because agent decisions under uncertainty must be evaluated in terms of expected outcomes, constraints, and feedback loops [39]. The proposed system uses reviewer feedback as a governance signal to improve prompt templates, retrieval filters, and remediation policies. The statistical evaluation should compare baseline and framework results across vulnerability categories, languages, repository sizes, and severity levels. Predictive validation of banking APIs demonstrates the importance of detecting workflow defects before they affect transaction processing [40]. In the same

way, the evaluation must determine whether AI-assisted remediation reduces risk before production release rather than merely improving scanner dashboards.

6. Static Analysis and Vulnerability Normalization

Static analysis tools are strongest when they provide precise localization and trace evidence. However, their outputs vary widely across tools. Some report line-level findings, while others report data-flow traces, dependency CVEs, infrastructure misconfiguration, or secret exposure. The framework therefore uses a normalized vulnerability object. Comparative analysis of Pivotal Cloud Foundry and OpenShift platforms shows that deployment environments differ in architecture, operational control, and portability [41]. Normalization allows findings from different tools and platforms to be represented consistently.

The vulnerability object contains the following fields: finding ID, tool name, severity, confidence, CWE, OWASP category, affected asset, vulnerable source, sink, path trace, dependency version, exploitability indicators, remediation template, owner, policy tags, and validation requirements. In user-facing enterprise systems, chatbot research highlights the importance of industry-specific constraints and ethical considerations when automation interacts with customers or business data [42]. In DevSecOps, policy tags perform a similar function by encoding industry-specific requirements such as HIPAA, PCI DSS, SOC 2, or internal data handling rules.

Static analysis is also used after remediation. The generated patch must be rescanned to verify that the original finding is resolved and that no new findings are introduced. SAP Fiori transition research shows that modernization efforts require guidelines, challenge analysis, and strategic implications because technical upgrades affect user experience and enterprise processes [43]. Secure remediation follows the same modernization principle: code change must be evaluated in the broader process context, not only as a local fix.

The framework also supports learned vulnerability prioritization. Historical findings, code churn, ownership, endpoint exposure, and dependency centrality can be used to prioritize findings. HPC-based machine learning research demonstrates that computationally intensive modeling can accelerate scientific discovery when simulations and analytics are integrated [44]. In enterprise DevSecOps, similar computational integration allows static findings to be enriched with historical repository analytics and runtime observability.

7. Retrieval-Augmented Security Knowledge Layer

The RAG layer is designed to provide grounded, auditable context for vulnerability explanation and remediation. It uses separate indexes for public security standards, internal policies, repository context, and approved historical patches. Each retrieved document is assigned a trust level, source type, freshness score, sensitivity level, and applicability score. Numerical methods research on convergence demonstrates that algorithmic stability depends on how iterative methods handle error and approximation [45]. RAG stability likewise depends on retrieval quality, prompt constraints, and rejection of low-trust context.

The public security index includes CWE definitions, OWASP vulnerability classes, secure coding standards, language-specific mitigation patterns, dependency advisories, and cloud security configuration guidance. The internal policy index includes approved encryption algorithms, logging restrictions, authentication requirements, dependency allowlists, secure API patterns, and branch protection rules. AI-powered root cause analysis and automated remediation research for multi-system data integrity shows that remediation becomes more effective when root causes are identified across system boundaries [46]. The RAG layer applies this idea by retrieving context across code, architecture, policy, and operational history.

Repository-specific retrieval is especially important because vulnerabilities are rarely fixed in isolation. A SQL injection patch may require an internal query builder rather than a generic ORM pattern. A cross-site scripting fix may require the organization's approved encoding utility. A broken access control issue may require a specific entitlement service. Converged AI architecture research for software lifecycle optimization and cybersecurity risk mitigation supports the idea that development, security, and innovation controls should be unified rather than separated [47].

The RAG layer includes prompt-injection defenses. It strips executable instructions from retrieved documentation unless the source is trusted policy content. It detects adversarial phrases such as "ignore previous instructions," "disable scanner," or "exfiltrate secrets." It also prevents retrieved issue descriptions or comments from overriding system policy. Reinforcement learning for dynamic service composition shows that adaptive decisions must respect constraints in changing network conditions [48]. Secure retrieval similarly adapts to context while respecting non-negotiable security rules.

8. Automated Vulnerability Remediation Workflow

The remediation workflow begins when the static analysis orchestrator identifies a finding above a configurable risk threshold. The vulnerability is normalized and enriched with repository metadata. The RAG layer retrieves relevant evidence. The LLM generates an explanation, patch plan, test plan, and candidate diff. The candidate patch is applied in an isolated branch and validated. If validation succeeds, the system opens a pull request with a structured remediation report. Intelligent ATM transaction validation research shows the value of real-time validation, fraud detection, and switching integrity in financial systems [49]. The remediation workflow applies the

same principle of real-time integrity checking to code changes.

For each patch, the system generates three classes of tests. The first class is a positive regression test that verifies the original intended behavior still works. The second class is a negative security test that verifies the exploit path is blocked. The third class is a boundary test that checks edge cases around input length, encoding, null values, authorization state, and malformed payloads. Deep learning work on patient journey mapping demonstrates that domain-specific workflows can benefit from predictive modeling when engagement paths are structured and measurable [50]. Secure remediation benefits when exploit paths are similarly structured and measurable.

The pull request includes a remediation summary, vulnerable path, retrieved evidence, patch rationale, files changed, tests generated, validation outcomes, residual risk, and rollback plan. The reviewer can accept, request changes, reject, or mark as false positive. Human-computer interaction research on emotion understanding and mental wellness assistance underscores that AI systems should be evaluated not only technically but also through user-centered interaction quality [51]. In DevSecOps, developer trust and reviewer usability are essential adoption factors.

The workflow also supports partial automation. Low-risk findings may receive an explanation and suggested patch without automatic branch creation. Medium-risk findings may receive generated tests and patch proposals. High-risk findings may require security engineer approval before any patch is applied. HANA in-memory computing research shows that high-performance systems require careful database-layer architecture and workload-specific design [52]. Similarly, remediation automation should be tuned to workload criticality and system risk.

9. Enterprise Implementation Considerations

Enterprise implementation requires integration with existing DevSecOps infrastructure. The framework can be deployed as a CI/CD extension connected to Git repositories, issue trackers, artifact repositories, secret managers, policy engines, and observability platforms. AI-driven quality engineering architecture for banking, API, and UAT ecosystems shows that end-to-end validation must span multiple quality gates and business-facing workflows [53]. SecureRAG-Remediate follows that approach by linking vulnerability remediation with testing, review, and deployment controls.

The system should support role-based access control. Developers may view findings and suggested patches for owned services. Security engineers may configure policy rules, approve high-risk remediation, and manage knowledge sources. Platform engineers may manage scanner execution and CI/CD integration. Audit teams may view remediation evidence and compliance reports. Adaptive intelligence in cloud observability and automated root cause analysis shows that enterprise systems benefit when monitoring and remediation are connected through unified intelligence [54].

For organizations using customer platforms, CRM systems, or regulated workflows, LLM remediation must respect data boundaries. Salesforce CRM research for real-time DeFi portfolio intelligence demonstrates that AI workflows may combine customer engagement, forecasting, and financial context [55]. When applied to DevSecOps, this means code and vulnerability context may contain sensitive business rules. The framework must support redaction, access filtering, and private model deployment where required.

Healthcare and clinical research modernization further illustrates the importance of combining technical transformation with data management controls [56]. In healthcare software, remediation artifacts may expose protected workflows, prescription rules, claims logic, or patient-related processing assumptions. Therefore, secure engineering pipelines must avoid sending sensitive code or data to external systems unless contractual, privacy, and compliance controls are in place.

From an engineering standpoint, the framework should be implemented incrementally. The first stage is read-only explanation, where the LLM summarizes findings and retrieves relevant policy. The second stage is patch suggestion without automatic application. The third stage is branch-based patch generation with automated tests. The fourth stage is policy-based autonomous pull request creation for selected low-risk vulnerability classes. Optimized backpropagation research reminds us that improvement often depends on controlled iteration and feedback rather than one-step automation [57].

10. Discussion

The proposed architecture changes the role of security tooling from detection-only to remediation-oriented governance. This shift is important because vulnerability backlogs often persist not because organizations lack scanners, but because they lack scalable remediation workflows. Decision intelligence methodology for agile software lifecycle governance supports the idea that software projects require architecture-centered decision frameworks that connect prediction, testing, and management [58]. SecureRAG-Remediate extends this approach to vulnerability remediation.

A major advantage of the framework is explainable remediation. Developers are more likely to accept patches when they understand the vulnerability, the exploit path, the selected fix pattern, and the test evidence. Ethical research on AI-based supply chain optimization warns that efficiency must be balanced with fairness and responsible constraints [59]. In DevSecOps, speed must be balanced with correctness, accountability, and secure-by-design principles.

Another advantage is organizational learning. Each accepted or rejected remediation becomes a training signal

for future retrieval and prompt templates. If reviewers consistently reject generic sanitization in favor of framework-specific encoders, the knowledge base can prioritize approved patterns. Predictive monitoring and error mitigation in change data capture pipelines demonstrate that historical errors can improve future monitoring and prevention [60]. The same feedback principle applies to vulnerability remediation.

The framework also supports multi-cloud and hybrid architectures. API architectures for multi-cloud enterprises show that centralized, distributed, and hybrid models each create different governance tradeoffs [61]. Secure remediation pipelines must work across these deployment patterns, especially when vulnerabilities affect shared APIs, gateway policies, service mesh rules, or identity propagation. RAG can retrieve architecture-specific controls so that patches align with the enterprise deployment model.

However, the framework has limitations. First, LLMs may generate plausible but incomplete fixes. Second, static analysis may miss vulnerabilities that require runtime context. Third, RAG can retrieve irrelevant or outdated documents. Fourth, generated tests may encode the same flawed assumption as the patch. Fifth, excessive automation may reduce developer security learning if teams accept patches passively. Fax-to-digital prescription automation research shows that cloud-native automation is valuable only when OCR, machine learning, and microservices are integrated with operational validation [62]. LLM DevSecOps requires the same integration discipline.

11. Challenges, Risk Controls, and Future Research

The first challenge is hallucinated remediation. An LLM may invent a nonexistent library function, misapply a security rule, or propose a patch that appears correct but fails under adversarial input. Fault prediction research using Random Forest, Logistic Regression, and KNeighbors demonstrates that model effectiveness varies across datasets and feature patterns [63]. This variability suggests that LLM remediation should be benchmarked across vulnerability classes rather than trusted uniformly.

The second challenge is overfitting to tests. A generated patch may pass existing tests while failing to preserve intended semantics. Therefore, future research should combine differential testing, symbolic execution, fuzzing, and human inspection. Monitoring and deployment optimization research using OpenShift and Helm highlights that deployment success depends on runtime observability and platform-specific validation [64]. Secure remediation should extend validation beyond CI into staged rollout monitoring.

The third challenge is knowledge poisoning. RAG stores may contain outdated documentation, insecure examples, or adversarial content. Future work should develop retrieval trust scoring, provenance verification, and policy-aware ranking. The fourth challenge is evaluation standardization. Research comparing machine learning techniques for software defect prediction shows that performance comparison requires consistent datasets, metrics, and experimental design [65]. LLM-driven remediation likewise needs shared benchmarks that measure exploit elimination, semantic preservation, and review burden.

Future research should investigate secure fine-tuning on approved organizational patches, privacy-preserving learning across repositories, vulnerability-specific prompt templates, multimodal architecture retrieval, agentic test generation, and post-deployment exploit monitoring. Another direction is integrating software bill of materials data with remediation reasoning so that dependency upgrades can be evaluated for compatibility and supply-chain risk. Finally, future work should explore developer education: instead of only generating patches, LLM systems should explain secure coding principles so that human capability improves over time.

12. Conclusion

This paper presented SecureRAG-Remediate, a reference architecture for LLM-driven DevSecOps that integrates static analysis, retrieval-augmented generation, controlled patch generation, automated validation, and human governance. The proposed framework treats LLMs as powerful but untrusted remediation assistants. It uses static analysis to identify evidence, RAG to ground reasoning in secure knowledge, validation gates to test generated patches, and audit trails to preserve accountability. The paper argued that the future of secure software engineering is not fully autonomous code repair, but governed AI-assisted remediation embedded inside mature DevSecOps pipelines. By combining secure software development practices, code-aware models, retrieval controls, enterprise policy, and measurable validation, organizations can reduce vulnerability backlogs while maintaining engineering discipline. The framework contributes a practical and research-oriented foundation for future empirical studies on secure, auditable, and trustworthy AI-assisted software remediation.

References

[1] M. Souppaya, K. Scarfone, and D. Dodson, “Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities,” NIST Special Publication 800-218, National Institute of Standards and Technology, Feb. 2022, doi: 10.6028/NIST.SP.800-218

[2] Sivva, S. D., Thalakanti, R. R., Bandari, S. S. G., & Yettapu, S. D. R. (2023). AI-Driven Decision Intelligence for Agile Software Lifecycle Governance: An Architecture-Centered Framework Integrating Machine Learning Defect Prediction and Automated Testing. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 167-172. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P118

[3] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” Advances in Neural Information Processing Systems, 2020

[4] Thalakanti, R. R., & Goud Bandari, S. S. . (2024). Intelligent Continuous Integration and Delivery for Banking Systems using Machine Learning Driven Risk Detection with Real World Deployment Evaluation. International Journal of AI, BigData, Computational and Management Studies, 5(4), 168-175. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I4P118

[5] Gudi, S. R. (2024). Design and Evaluation of Secure Microservices Architecture for HIPAA-Compliant Prescription Processing on AWS and OpenShift. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(2), 144-149. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I2P116

[6] Z. Feng et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1536–1547, 2020.

[7] S. K. Gunda, "Automatic Software Vulnerabilty Detection Using Code Metrics and Feature Extraction," 2025 2nd International Conference on Multidisciplinary Research and Innovations in Engineering (MRIE), Gurugram, India, 2025, pp. 115–120, https://doi.org/10.1109/MRIE66930.2025.11156601

[8] Bandari, S. S. G. ., Sivva, S. D. ., & Thalakanti, R. R. (2024). Regulatory Grade Fraud Detection using Explainable Artificial Intelligence with Auditable Decision Pathways and Empirical Validation on Banking Data. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(3), 139-147. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I3P115

[9] S. Yalamati, "Energy-Efficient Task Offloading in Multi-Tenant Edge Clouds," 2026 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2026, pp. 379–384, doi: 10.1109/ICESIC67389.2026.11496473

[10] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications 2025," OWASP GenAI Security Project, 2025.

[11] Gudi, S. R. (2023). Enhancing Reliability in Java Enterprise Systems through Comparative Analysis of Automated Testing Frameworks. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 151-160. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P115

[12] A. K. K. V. Alluri, "A Systematic Study of Machine Learning Frameworks Enabling Scalable Secure and Explainable Artificial Intelligence in Salesforce CRM Platforms," 2026 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2026, pp. 396–401, doi: 10.1109/ICESIC67389.2026.11496486

[13] D. Guo et al., "GraphCodeBERT: Pre-training Code Representations with Data Flow," International Conference on Learning Representations, 2021.

[14] S. K. Gunda, "Comparative Analysis of Machine Learning Models for Software Defect Prediction," 2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 2024, pp. 1–6, https://doi.org/10.1109/ICPECTS62210.2024.10780167

[15] R. R. Thalakanti, "Formalizing feature model integrity: a typing system and refactoring approaches for improving software product line design," International Conference on Advancing Technology in Engineering and Science (ICATES 2025), Mumbai, India, 2026, pp. 710–717, doi: 10.1049/icp.2025.4792

[16] C. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" 2024.

[17] S. R. Gudi, "Ensuring Secure and Compliant Fax Communication: Anomaly Detection and Encryption Strategies for Data in Transit," 2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Tirupur, India, 2025, pp. 786–791, https://doi.org/10.1109/ICIMIA67127.2025.11200537

[18] S. S. G. Bandari, S. Banda, and S. Naik, "Machine Learning (ML) based Anomaly Detection in Insurance Industries," Journal of Information Systems Engineering and Management, vol. 10, no. 32s, 2026, https://doi.org/10.52783/jisem.v10i32s.5182

[19] Gunda, S. K. G. (2023). The Future of Software Development and the Expanding Role of ML Models. International Journal of Emerging Research in Engineering and Technology, 4(2), 126-129. https://doi.org/10.63282/3050-922X.IJERET-V4I2P113

[20] V. K. R. Mittamidi, "An Automated AI-Driven Monitoring and Observability Framework for Cloud-Based Data Pipelines by Software Defect Prediction Research," International Journal of Multidisciplinary Evolutionary Research, vol. 5, no. 1, pp. 109–112, 2024, doi: 10.54660/IJMER.2024.5.1.109-112

[21] OWASP Foundation, "LLM08:2025 Vector and Embedding Weaknesses," OWASP GenAI Security Project, 2025.

[22] S. Yalamati, "AI-Augmented Service Fabric for Adaptive Resource Management in Cloud Environments," 2025 5th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS), Erode, India, 2025, pp. 963–968, doi: 10.1109/ICUIS67429.2025.11380548

[23] S. R. Gudi, "Deconstructing Monoliths: A Fault-Aware Transition to Microservices with Gateway Optimization using Spring Cloud," 2025 6th International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2025, pp. 815–820, https://doi.org/10.1109/ICESC65114.2025.11212326.

[24] Kishore Varma Alluri, A. K. (2026). Governed Agentic AI for Salesforce CRM Platforms: A Reference Architecture for Data Grounding, Decision Intelligence, Trust Controls, and Lifecycle Reliability. International Journal of Emerging Trends in Computer Science and Information Technology, 7(1), 374-382. https://doi.org/10.63282/3050-9246.IJETCSIT-V7I1P153

[25] E. Basic and A. Giaretta, "Large Language Models and Code Security: A Systematic Literature Review," arXiv:2412.15004, 2024.

[26] S. K. Gunda, "A Hybrid Deep Learning Model for Software Fault Prediction Using CNN, LSTM, and Dense Layers," in Internet and Modern Society, IMS 2025, Communications in Computer and Information Science, vol. 2672, Springer, Cham, 2026, https://doi.org/10.1007/978-3-032-05144-8_21

[27] Thalakanti, R. R. ., Goud Bandari, S. S., & Sivva, S. D. . (2024). Federated Learning for Privacy Preserving Fraud Detection across Financial Institutions: Architecture Protocols and Operational Governance. International Journal of Emerging Research in Engineering and Technology, 5(2), 108-114. https://doi.org/10.63282/3050-922X.IJERET-V5I2P111

[28] S. R. Gudi, "Enhancing optical character recognition (OCR) accuracy in healthcare prescription processing using artificial neural networks," European Journal of Artificial Intelligence and Machine Learning, vol. 4, no. 6, 2025, https://doi.org/10.24018/ejai.2025.4.6.79

[29] T. Raikar, "Preserving the clean core principles in SAP systems: Design strategies for integrating AI," 2026 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2026, pp. 1036–1041, doi: 10.1109/ICESIC67389.2026.11496501

[30] B. S. M. Rao and S. S. G. Bandari, "Replacing AI Agents for Backend," International Journal of Scientific Research in Engineering and Management, https://doi.org/10.55041/IJSREM.NCFT011

[31] S. Yalamati, "Sparse Matrix Factorization for Scalable Machine Learning in Cloud Environments," 2025 International Conference on NexGen Networks and Cybernetics (IC2NC), Erode, India, 2025, pp. 333–338, doi: 10.1109/IC2NC67409.2025.11376338

[32] Gunda, S. K. (2025). AI-Enhanced API Reliability Testing for Digital Banking: Improving Accuracy, Resilience, and Integrity in Financial Transaction Processing. International Journal of Emerging Trends in Computer Science and Information Technology, 6(2), 136-143. https://doi.org/10.63282/3050-9246.IJETCSIT-V6I2P116

[33] S. D. Sivva, "An end-to-end AI-based systems engineering paradigm for lifecycle governance, predictive quality assurance, automation economics, and cybersecurity intelligence," Journal of Frontiers in Multidisciplinary Research, vol. 4, no. 1, pp. 600–604, 2023, https://doi.org/10.54660/.JFMR.2023.4.1.600-604

[34] Gudi, S. R. (2024). Leveraging Predictive Analytics and Redis-Backed Caching to Optimize Specialty Medication Fulfillment and Pharmacy Inventory Management. International Journal of AI, BigData, Computational and Management Studies, 5(3), 155-160. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I3P116

[35] R. R. Thalakanti, "Optimizing Neural Network Architecture for Binary Classification Using Evolutionary Algorithms," 2025 International Conference on Electronics and Computing, Communication Networking Automation Technologies (ICEC2NT), Pune, India, 2025, pp. 1–6, doi: 10.1109/ICEC2NT65402.2025.11380048

[36] A. K. K. V. Alluri and S. Barde, "AI Powered Decision Intelligence Frameworks for Predictive and Prescriptive Business Optimization in Salesforce Enterprise Platforms," 2026 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2026, pp. 438–443, doi: 10.1109/ICESIC67389.2026.11496409

[37] S. K. Gunda, "An exploration of adaptive ensemble approaches in software fault detection: Balancing accuracy and robustness," AIP Conference Proceedings, vol. 3345, no. 1, p. 020211, Jan. 7, 2026, https://doi.org/10.1063/5.0298093

[38] N. Mutyam, "Graph-based modeling of service dependencies for predicting failure propagation in distributed systems," International Journal of Multidisciplinary Evolutionary Research, vol. 5, no. 1, pp. 113–116, 2024, https://doi.org/10.54660/IJMER.2024.5.1.113-116

[39] S. Yalamati, "Probabilistic Reasoning in Multi-Agent Reinforcement Learning Systems," 2025 International Conference on NexGen Networks and Cybernetics (IC2NC), Erode, India, 2025, pp. 707–712, doi: 10.1109/IC2NC67409.2025.11376303

[40] Gunda, S. K. (2025). Predictive Validation of Banking APIs and Transaction Workflows Using Machine Learning-Based Defect Detection Model. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 6(1), 284-292. https://doi.org/10.63282/3050-9262.IJAIDSML-V6I1P133

[41] S. R. Gudi, "A Comparative Analysis of Pivotal Cloud Foundry and OpenShift Cloud Platforms," The American Journal of Applied Sciences, vol. 7, no. 07, pp. 20–29, 2025, https://doi.org/10.37547/tajas/Volume07Issue07-03

[42] S. Naik, P. Aitharaju, and S. S. Bandari, "AI Chatbots in Enterprise Solutions: Transforming Customer Support, Industry-Specific Challenges and Ethical Considerations," Glovento Journal of Integrated Studies, 2025.

[43] T. Raikar and V. Apelagunta, "Implementing SAP Fiori in S/4HANA Transitions: Key Guidelines, Challenges, Strategic Implications, AI Integration Recommendations," Journal of Engineering Research and Sciences, vol. 4, no. 11, pp. 1–9, 2025, https://doi.org/10.55708/js0411001

[44] S. K. Gunda, "Accelerating Scientific Discovery With Machine Learning and HPC-Based Simulations," in Integrating Machine Learning Into HPC-Based Simulations and Analytics, B. Ben Youssef and M. Ben Ismail, Eds., IGI Global Scientific Publishing, pp. 229–252, 2025, https://doi.org/10.4018/978-1-6684-3795-7.ch009

[45] R. R. Thalakanti, "Convergence Analysis and Implementation of Linear Multistep Methods for Solving Ordinary Differential Equations," 2025 2nd Asian Conference on Intelligent Technologies (ACOIT),

KOLAR, India, 2025, pp. 1–18, doi: 10.1109/ACOIT66109.2025.11436783

[46] Reddy Mittamidi, V. K. (2025). AI/ML Powered Intelligent Root Cause Analysis and Automated Remediation for Multi System Data Integrity Issues. International Journal of AI, BigData, Computational and Management Studies, 6(4), 133-141. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I4P115

[47] M. Balerao, "A converged artificial intelligence architecture for innovation, software lifecycle optimization, and cybersecurity risk mitigation," International Journal of Multidisciplinary Futuristic Development, vol. 4, no. 1, pp. 117–120, 2023, https://doi.org/10.54660/IJMFD.2023.4.1.117-120.

[48] S. Yalamati, "Reinforcement Learning for Dynamic Service Composition in Edge Networks," 2025 4th International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 2025, pp. 1158–1163, doi: 10.1109/ICAAIC64647.2025.11330768.

[49] Gunda, S. K. (2024). An Intelligent AI-Driven Framework for Real-Time ATM Transaction Validation, Fraud Detection and Financial Switching Integrity. International Journal of Emerging Research in Engineering and Technology, 5(4), 180-191. https://doi.org/10.63282/3050-922X.IJERET-V5I4P119

[50] Kishore Varma Alluri, A. K. (2025). Using Salesforce CRM and Deep Learning (CNN) Techniques to Improve Patient Journey Mapping and Engagement in Small and Medium Healthcare Organizations. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 6(4), 101-109. https://doi.org/10.63282/3050-9262.IJAIDSML-V6I4P115

[51] Krishna, G. V., Reddy, B. D., & Vrindaa, T. (2025). EmoVision: An Intelligent Deep Learning Framework for Emotion Understanding and Mental Wellness Assistance in Human Computer Interaction. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 6(4), 14-20. https://doi.org/10.63282/3050-9262.IJAIDSML-V6I4P103

[52] T. Raikar, "High-Performance In-Memory Computing: A Research Study on SAP S/4 HANA Database Layer," American Journal of Technology, vol. 4, no. 2, pp. 93–113, 2025, https://doi.org/10.58425/ajt.v4i2.449.

[53] Gunda, S. K. (2025). A Scalable AI-Driven Quality Engineering Architecture for End-To-End Validation of Core Banking, API, and UAT Ecosystems. American International Journal of Computer Science and Technology, 7(6), 126-138. https://doi.org/10.63282/3117-5481/AIJCST-V7I6P113

[54] Manga, I., Sivva, S. D. ., & Manga, V. K. (2024). The Adaptive Intelligence in Cloud Systems: A Unified Architecture for AI Enhanced Observability and Automated Root Cause Analysis. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(1), 160-166. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I1P115

[55] Kishore Varma Alluri, A. K. . (2025). Salesforce CRM Framework for Real Time DeFi Portfolio Intelligence and Customer Engagement Forecasting in Web3 Based Decentralized Finance Ecosystems Using ML Techniques. International Journal of AI, BigData, Computational and Management Studies, 6(4), 99-107. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I4P111

[56] M. Ukey, S. R. Abbidi, T. K. Kota, T. Raikar, M. Mallepati, and P. J. Adinarayana, "Digital transformation in healthcare: Integrating clinical research with data management technologies," in Proc. 2026 6th International Conference on Recent Trends in Computer Science and Technology (ICRTCST), Jamshedpur, India, 2026, pp. 886–891, doi: 10.1109/ICRTCST68392.2026.11545210.

[57] R. R. Thalakanti, "Enhancing Convergence in Fully Connected Neural Networks via Optimized Backpropagation," 2025 2nd International Conference on Computing and Data Science (ICCDS), Chennai, India, 2025, pp. 1–6, doi: 10.1109/ICCDS64403.2025.11209625.

[58] Gunda, S. K., Yettapu, S. D. R., Bodakunti, S., & Bikki, S. B. (2023). Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(1), 102-108. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P112

[59] T. Raikar, "Ethics of AI-based supply chain optimization: A better balance between efficiency and fairness," Mar. 2026, https://doi.org/10.55670/fpll.futech.5.2.26.

[60] Reddy Mittamidi, V. K. (2025). Leveraging AI and ML for Predictive Monitoring and Error Mitigation in Change Data Capture Pipelines. International Journal of Emerging Trends in Computer Science and Information Technology, 6(3), 104-111. https://doi.org/10.63282/3050-9246.IJETCSIT-V6I3P116

[61] AI-Driven API Architectures for Multi-Cloud Enterprises: A Comparative Study of Centralized, Distributed, and Hybrid Deployment Models. (2026). International Journal of Computer Science and Engineering Innovations, 2(1), 60-67. https://doi.org/10.64137/31079458/IJCSEI-V2I1P108

[62] Gudi, S. R. (2024). AI-Driven Fax-to-Digital Prescription Automation: A Cloud-Native Framework Using OCR, Machine Learning, and Microservices for Pharmacy Operations. International Journal of Emerging Research in Engineering and Technology, 5(1), 111-116. https://doi.org/10.63282/3050-922X.IJERET-V5I1P113

[63] S. K. Gunda, "Fault Prediction Unveiled: Analyzing the Effectiveness of Random Forest, Logistic Regression, and KNeighbors," 2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), Erode, India, 2024, pp. 107–113, https://doi.org/10.1109/ICSSAS64001.2024.10760620.

[64] S. R. Gudi, "Monitoring and Deployment Optimization in Cloud-Native Systems: A Comparative Study Using OpenShift and Helm," 2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Tirupur, India, 2025, pp. 792–797, https://doi.org/10.1109/ICIMIA67127.2025.11200594.

[65] S. K. Gunda, "Analyzing Machine Learning Techniques for Software Defect Prediction: A Comprehensive Performance Comparison," 2024 Asian Conference on Intelligent Technologies (ACOIT), KOLAR, India, 2024, pp. 1–5, https://doi.org/10.1109/ACOIT62457.2024.10939610.