International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 7 Issue 2 PP 282-291, 2026
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V7I2P134

Original Article

Agentic AI for Autonomous Software Maintenance: A Retrieval-Augmented Multi-Agent Framework for Bug Localization, Patch Generation, and Regression Validation

Dr. Swathi .J .N[1], Dr. S. L. Aarthy[2], Dr. Ananda Kumar .S[3], Dr. Debi Prasanna Acharjya[4]
[1, 2, 3, 4]Professor, CS, Computational Intelligence, Vellore Institute of Technology, Vellore, Tamilnadu, India.

Received On: 18/04/2026 ***Revised On: 17/05/2026*** ***Accepted On: 25/05/2026*** ***Published On: 01/06/2026***

Abstract *- Autonomous software maintenance is becoming a critical research direction as modern codebases grow beyond the practical reach of manual debugging, release governance, and regression validation. Large language models have demonstrated strong code-generation capabilities, but repository-level maintenance remains difficult because defects frequently arise from interactions among multiple files, hidden dependencies, runtime configurations, and incomplete issue reports. This paper proposes AutoMaint-RAG, a retrieval-augmented multi-agent framework for autonomous software maintenance. The framework coordinates specialized agents for issue interpretation, repository retrieval, fault localization, patch synthesis, static analysis, test generation, regression execution, and release-risk governance. Unlike single-agent code-generation workflows, AutoMaint-RAG treats maintenance as a controlled software-engineering process in which every generated patch must be grounded in repository evidence, ranked suspicious locations, executable test feedback, and auditable decision records. The proposed framework integrates hybrid retrieval over code, commit history, test traces, dependency graphs, and operational telemetry; combines spectrum-based fault localization with semantic ranking; and applies iterative patch refinement through execution-guided feedback. A reproducible validation protocol is defined using repository-level issue benchmarks, mutation-based regression assessment, and industrial-style CI/CD gates. The paper contributes a formal architecture, agent-interaction protocol, patch-risk scoring model, evaluation design, and governance layer for deploying agentic AI in safety-sensitive software maintenance environments. The central argument is that autonomous maintenance should not be reduced to prompt-based patch generation; rather, it requires an engineered socio-technical pipeline that constrains language-model creativity through retrieval grounding, tool execution, regression evidence, and human-review escalation. The proposed framework advances a research agenda for maintainable, auditable, and operationally reliable AI agents in software lifecycle management.*

Keywords *- Agentic AI, Autonomous Software Maintenance, Retrieval-Augmented Generation, Bug Localization, Automated Program Repair, Regression Validation, Multi-Agent Systems, CI/CD Governance.*

1. Introduction

Recent repository-level benchmarks have demonstrated that real-world software-engineering issues require models to edit codebases rather than merely complete isolated functions [1]. This shift is important because repository maintenance involves cross-file reasoning, build-system awareness, dependency interpretation, and test-oracle alignment. A model may generate syntactically plausible code and still fail because it modifies the wrong abstraction, ignores a hidden invariant, or overfits to a narrow failing test. Therefore, autonomous maintenance requires an architecture that can retrieve evidence, execute tools, inspect failures, and refine its actions through feedback rather than producing a single unconstrained patch.

Agentic AI provides a promising foundation for this problem because agents can combine reasoning, action, memory, and tool use in iterative workflows. The ReAct paradigm showed that language models can alternate between reasoning and environment interaction, making them suitable for tasks that require intermediate observations and decisions [3]. However, software maintenance differs from many general agent tasks because the environment is unforgiving: a patch either compiles, passes tests, preserves behavior, and meets review standards, or it fails. Consequently, a maintenance agent must be evaluated not only by text quality but also by executable correctness, traceability, and risk reduction.

Existing AI-driven software lifecycle governance research emphasizes the importance of linking defect prediction, automated testing, and decision intelligence into architecture-centered lifecycle management [2]. In this paper, that view is extended to autonomous maintenance by treating the agent as one component in a governed pipeline rather than as an unconstrained code assistant. The proposed AutoMaint-RAG framework coordinates multiple agents around a shared evidence store and enforces explicit gates for localization confidence, patch minimality, regression coverage, security review, and release readiness.

The motivation is reinforced by industrial CI/CD contexts in which risk-based deployment decisions are increasingly supported by machine-learning signals [4]. Yet, most current agentic coding workflows focus on patch generation and issue resolution, while fewer address operational governance after the patch is produced. In regulated or high-availability environments, a generated fix is not useful unless it can be justified, reproduced, tested, and rolled back. Thus, autonomous maintenance should be designed as a closed-loop engineering system with retrieval, localization, synthesis, validation, and governance phases.

Retrieval-augmented generation is central to the proposed approach because it mitigates the mismatch between model context windows and repository-scale codebases. RAG originally combined parametric language-model knowledge with non-parametric retrieval to improve factual grounding [5]. In software maintenance, the retrieved corpus is not external encyclopedic text but repository-specific evidence: source files, tests, stack traces, issue reports, recent commits, architectural decision records, API contracts, configuration files, and production telemetry. This repository memory enables an agent to ground its patch in the actual system rather than in generic programming knowledge.

This paper makes five contributions. First, it defines a multi-agent architecture for autonomous maintenance that decomposes work into interpretable roles. Second, it proposes a hybrid retrieval mechanism that integrates semantic code search, dependency graphs, test traces, and operational signals. Third, it introduces a fault-localization and patch-generation workflow that combines ranked suspicious locations with execution-guided refinement. Fourth, it defines a regression-validation protocol that uses existing tests, generated tests, mutation checks, and risk scoring. Fifth, it presents a governance model for human escalation, auditability, and CI/CD integration.

2. Research Problem and Design Requirements

The research problem addressed in this paper is: How can autonomous agents localize bugs, generate patches, and validate regressions in repository-scale software systems while remaining grounded, auditable, and safe for CI/CD integration? This problem has four subproblems. The first is context selection: the system must decide which files, tests, logs, and historical artifacts are relevant. The second is localization: the system must rank the program locations most likely to contain the defect. The third is synthesis: the system must generate a patch that addresses the root cause without introducing unrelated changes. The fourth is validation: the system must determine whether the patch preserves existing behavior and satisfies the issue requirement.

Secure microservice research demonstrates that maintainability and compliance depend on explicit architectural boundaries, controlled interfaces, and deployment discipline [6]. These same principles apply to agentic maintenance. If an agent modifies a service without understanding its interface contracts or compliance constraints, it can create latent failures that pass local tests but fail in production. Therefore, the framework must reason about system structure, not just source-code snippets. Enterprise SAP transition studies similarly emphasize user-experience, integration, and change-management constraints in S/4HANA modernization [9], while clean-core integration research highlights the need to preserve architectural boundaries when AI capabilities are introduced into SAP systems [10].

SWE-agent established the importance of agent-computer interfaces for repository navigation, editing, and test execution [7]. AutoMaint-RAG builds on that insight but introduces a stronger separation of duties. Instead of one agent attempting to perform every action, specialized agents communicate through structured artifacts: issue hypotheses, retrieved evidence packs, suspicious-location rankings, candidate patches, test reports, and risk summaries. This decomposition improves observability and allows organizations to apply different trust thresholds to different stages.

Enterprise decision-intelligence frameworks also indicate that AI systems become operationally useful when model outputs are tied to business policies, governance rules, and traceable decisions [8]. For software maintenance, this means that the system should not merely output a patch; it should output a maintenance case file. The case file records why the bug was localized to specific files, which evidence supported the change, which tests were executed, what risk remains, and whether human review is required.

3. Related Work and Research Gap

Automated program repair has long studied methods for generating patches from failing tests and correctness oracles. Classical APR research defines the repair problem as producing a patch that transforms a buggy program into one that satisfies a test suite [33]. However, traditional APR frequently depends on limited search spaces, templates, or test suites that may not fully represent intended behavior. In contrast, large language models can synthesize more diverse patches, but they also introduce risks of hallucination, overfitting, and unjustified edits.

RAG-enhanced repair approaches have begun to address grounding by retrieving relevant external or contextual evidence for bug localization and repair [12]. Nevertheless, many RAG workflows retrieve text passages without deeply integrating execution traces, dependency relations, or regression evidence. AutoMaint-RAG treats retrieval as a multi-modal software-engineering process, combining semantic search with structural and dynamic signals. This distinction matters because maintenance failures often arise from code interactions that are invisible in plain-text similarity.

Repository-level code generation research has shown that real-world generation requires long-range dependency awareness, global consistency, and retrieval over large

codebases [37]. Yet, code-completion research and maintenance research are not identical. Completion predicts missing code, whereas maintenance must preserve existing behavior while changing faulty behavior. The regression objective makes maintenance stricter than many generation tasks.

AutoCodeRover demonstrated that repository-level program improvement benefits from program-structure-aware search and fault-localization cues [19]. This paper extends that direction by proposing a full multi-agent lifecycle, including validation and release governance. Similarly, iterative code-generation flows such as AlphaCodium emphasize the value of generating, running, and fixing code against tests [22]. AutoMaint-RAG adapts that execution loop to maintenance by adding fault-localization agents, regression-risk agents, and governance artifacts.

Reflection-based language-agent research shows that agents can improve behavior by storing feedback in memory and using it to guide subsequent attempts [16]. AutoMaint-RAG uses a related principle, but the memory is constrained: reflections must be linked to executable evidence such as failing stack traces, test output, static-analysis warnings, or reviewer comments. This prevents the system from accumulating ungrounded self-justifications.

Self-refinement techniques further show that iterative feedback can improve model outputs without changing model weights [40]. In software maintenance, however, feedback must be operational rather than merely linguistic. The strongest feedback signals come from compilers, unit tests, integration tests, linters, static analyzers, mutation tests, and runtime traces. Therefore, AutoMaint-RAG makes tool feedback first-class in the agent protocol.

4. AutoMaint-RAG Framework

AutoMaint-RAG consists of eight cooperating agents: the Issue Triage Agent, Retrieval Agent, Fault Localization Agent, Patch Planning Agent, Patch Generation Agent, Test Synthesis Agent, Regression Validation Agent, and Governance Agent. Each agent has a bounded responsibility and produces structured outputs that can be inspected independently.

The Issue Triage Agent converts a raw issue report into a maintenance specification. It extracts the observed failure, expected behavior, affected component, reproduction steps, error messages, environmental assumptions, and uncertainty markers. If the report is incomplete, the agent generates a clarification checklist but continues with available evidence. Research on AI-driven lifecycle governance supports this emphasis on converting unstructured lifecycle artifacts into decision-ready information [47].

The Retrieval Agent builds an evidence pack. It combines lexical search, embedding search, call-graph expansion, dependency mapping, commit-history mining, and test-trace retrieval. Graph-based service dependency modeling is especially useful because failures in distributed systems often propagate across services rather than remaining local to a single function [17]. The Retrieval Agent therefore retrieves not only code files but also upstream callers, downstream consumers, configuration dependencies, and related tests.

The Fault Localization Agent ranks candidate locations using a hybrid score. The score combines semantic similarity to the issue, historical defect density, stack-trace proximity, dependency centrality, and spectrum-based suspiciousness when failing and passing test coverage are available. Spectrum-based fault localization provides a principled way to rank suspicious program elements from test execution behavior [25]. The agent emits a ranked list of functions, methods, files, and configuration elements with confidence values and evidence links.

The Patch Planning Agent converts localization results into a minimal change plan. The plan specifies which files may be edited, which invariants must be preserved, which tests should be run, and which failure mode the patch is intended to resolve. This prevents the Patch Generation Agent from making broad, opportunistic edits. Research on feature-model integrity and refactoring highlights the importance of maintaining structural correctness when software variants or product-line constraints exist [42].

The Patch Generation Agent produces candidate patches using the change plan and evidence pack. It is prohibited from editing files outside the approved scope unless it requests a scope expansion. This constraint is important for auditability. The agent also generates a rationale that maps each changed line to a failure hypothesis. Prior work on ML-based defect prediction supports the value of connecting defect signals to engineering decisions rather than treating predictions as isolated scores [30].

The Test Synthesis Agent creates additional tests when the existing test suite does not sufficiently cover the reported behavior. These tests include reproduction tests, boundary tests, contract tests, and regression tests for nearby functionality. Comparative analysis of automated testing frameworks indicates that tool choice affects reliability outcomes in enterprise systems [18]. AutoMaint-RAG therefore records which test framework was used, which assertions were added, and whether generated tests are retained or used only for validation.

The Regression Validation Agent executes a staged validation plan. The first stage runs targeted failing tests. The second stage runs related unit and integration tests. The third stage runs static analysis, linting, type checking, and security scans. The fourth stage optionally runs mutation testing or differential tests to detect overfitting. Cloud-native deployment research shows that deployment optimization depends on test evidence, observability, and platform-specific validation [35].

The Governance Agent produces a maintenance decision. It calculates a patch-risk score from localization confidence, patch size, test coverage, modified criticality, security impact, and historical failure rates. Banking API validation work shows why critical transaction systems require stronger validation thresholds than ordinary application modules [11]. If risk exceeds a policy threshold, the Governance Agent escalates the patch for human review rather than approving autonomous merge.

5. Retrieval and Localization Method

The retrieval layer is organized around five indices. The first is a code-text index built from functions, classes, comments, and docstrings. The second is a structural index built from call graphs, import graphs, and service dependencies. The third is a test index linking tests to covered files, fixtures, and assertions. The fourth is a temporal index built from commits, pull requests, issue history, and recent dependency updates. The fifth is an operational index built from logs, traces, metrics, and incident records.

Sparse matrix methods for scalable machine learning are relevant because repository retrieval often produces high-dimensional representations over tokens, paths, dependencies, and historical features [13]. AutoMaint-RAG uses sparse lexical retrieval for exact identifiers and dense embeddings for semantic similarity. Exact identifier retrieval is essential in software because small tokens such as method names, feature flags, and configuration keys often determine correctness. For SAP-oriented enterprise repositories, the database layer can also shape retrieval and validation requirements because S/4HANA relies on high-performance in-memory data processing models [14].

The retrieval score for artifact (a) is defined as a weighted composition of semantic similarity, lexical overlap, graph proximity, temporal recency, and execution relevance. The score is not used as a final answer; it is used to construct a candidate evidence pack for agent reasoning. This design prevents retrieval from becoming a brittle single-ranking mechanism.

Fault localization then combines retrieval evidence with dynamic execution evidence. If failing tests are available, the system instruments execution to collect coverage. If stack traces are available, the top frames receive higher priority. If no test fails, the system uses issue text, dependency graphs, and historical defect predictors to estimate candidate files. Predictive monitoring research in change-data-capture pipelines shows that runtime signals can identify error-prone data paths before failures become visible in downstream systems [20].

The localization score is intentionally interpretable. For each candidate location, the system stores the top evidence items, including issue phrases, retrieved code snippets, stack frames, coverage statistics, and commit links. This design allows human reviewers to inspect whether the agent localized the bug for plausible reasons. Anomaly detection and decision-path auditing research in regulated financial systems supports the need for explainable decision pathways in AI-supported operational decisions [15].

6. Patch Generation and Refinement Protocol

Patch generation follows a constrained iterative protocol. In iteration zero, the Patch Planning Agent defines the expected behavioral change. In iteration one, the Patch Generation Agent proposes the smallest patch consistent with the plan. The Regression Validation Agent then runs targeted tests and returns structured failures. The Patch Generation Agent may revise the patch, but each revision must explain how it responds to test feedback.

This protocol is designed to reduce overfitting. Automated repair systems may generate patches that satisfy visible tests while violating hidden requirements [68]. AutoMaint-RAG counters this risk through patch minimality constraints, generated regression tests, mutation checks, and reviewer escalation. A patch that deletes a failing assertion, broadens an exception handler, or disables a feature flag without justification is marked as suspicious.

API architecture research across centralized, distributed, and hybrid deployment models indicates that patch effects depend strongly on deployment topology [39]. For this reason, AutoMaint-RAG distinguishes local code correctness from system-level safety. A patch that passes unit tests may still affect API latency, serialization compatibility, cache invalidation, or downstream schema expectations. The Governance Agent therefore requires additional validation for public APIs, data pipelines, authentication flows, payment workflows, and safety-critical modules.

For cloud-native systems, the framework integrates with containerized test environments and deployment descriptors. Research on microservice migration and gateway optimization shows that architectural transitions can expose hidden failure modes if routing, service discovery, and gateway constraints are not validated [49]. AutoMaint-RAG therefore retrieves deployment manifests, gateway rules, and configuration files whenever the suspicious code belongs to a service boundary. In data-intensive systems, the patch workflow also inspects schema migrations, CDC jobs, and cache layers. Intelligent root-cause analysis research for multi-system data integrity problems emphasizes that failures may originate from interactions among systems rather than from a single faulty function [67]. AutoMaint-RAG reflects this by allowing the Retrieval Agent to expand from code to data contracts, transformation jobs, and lineage metadata.

7. Regression Validation and Release Governance

Regression validation is divided into technical validation and release governance. Technical validation answers whether the patch appears correct under available tests and analyses. Release governance answers whether the patch is safe to merge or deploy under organizational policy.

The validation pipeline begins with reproduction. If the original failure can be reproduced, the system records the failing command, environment, stack trace, and baseline output. If reproduction fails, the framework marks the issue as non-reproducible and reduces confidence. Then, after patch generation, the same command is rerun to confirm behavioral change. Research on predictive validation of banking APIs reinforces the importance of validating transaction workflows against realistic execution paths [51].

Next, the system runs regression tests selected by dependency proximity. Rather than executing the entire test suite first, AutoMaint-RAG selects tests linked to modified files, callers, configuration changes, and historical co-change clusters. If these pass, broader suites may be executed depending on risk. Reinforcement-learning approaches for dynamic service composition suggest that adaptive selection can improve resource use when service conditions vary [31].

Generated tests are handled carefully. A generated test may be useful for reproducing the issue, but it should not be treated as an authoritative oracle without review. The system labels generated tests as provisional until they are accepted by a maintainer. This avoids the risk that the agent writes a test that merely matches its own patch. Research on AI chatbots in enterprise solutions similarly warns that domain-specific AI systems require ethical and operational controls rather than blind automation [23]. AI optimization research in supply-chain contexts further reinforces the need to balance efficiency gains with fairness constraints [24].

Security and compliance validation are also integrated. If the patch touches authentication, authorization, data serialization, encryption, or personally identifiable information, the framework runs additional security checks. Work on secure fax communication and encryption strategies illustrates how domain-specific data-in-transit requirements can shape validation gates [59]. In healthcare or financial systems, an otherwise correct patch may still be unacceptable if it weakens data protection.

Finally, the Governance Agent creates a patch dossier. The dossier includes issue summary, evidence pack, localization ranking, patch diff, test report, generated tests, static-analysis output, security flags, rollback notes, and recommended merge decision. Governed agentic AI research for enterprise CRM platforms highlights the need for trust controls, grounding, lifecycle reliability, and decision governance [53]. AutoMaint-RAG applies the same principles to software maintenance.

8. Evaluation Protocol

A rigorous evaluation of AutoMaint-RAG should use repository-level maintenance benchmarks, controlled enterprise-style testbeds, and ablation studies. The primary benchmark category consists of real issue-to-patch datasets in which a system receives an issue report and codebase and must produce a patch validated by tests. SWE-bench is appropriate because it evaluates language models on real GitHub issues and corresponding pull-request fixes [1].

The experimental protocol should compare AutoMaint-RAG with four baselines: direct LLM patch generation, single-agent tool-using repair, retrieval-only repair, and localization-only repair. The direct baseline receives the issue and limited code context. The single-agent baseline can browse files and run tests. The retrieval-only baseline receives an evidence pack but no specialized localization or governance agents. The localization-only baseline receives suspicious locations but no iterative validation loop.

Metrics should include resolved issue rate, first-patch pass rate, number of iterations, patch size, localization precision, regression pass rate, generated-test utility, static-analysis warnings, and human-review escalation rate. Cost and latency should also be measured because autonomous maintenance must be practical for CI/CD use. AI-augmented service-fabric research similarly emphasizes adaptive resource management in cloud environments [55].

Ablation studies should remove one component at a time. Removing structural retrieval tests whether semantic retrieval alone is sufficient. Removing dynamic fault localization tests whether coverage and stack traces improve localization. Removing generated regression tests evaluates overfitting risk. Removing governance tests whether autonomous merge recommendations become unsafe. Probabilistic reasoning in multi-agent reinforcement learning provides a useful conceptual basis for analyzing uncertainty across interacting decision agents [46].

A separate industrial validation track should use anonymized enterprise repositories or synthetic replicas with realistic build systems, service dependencies, and test suites. Insurance anomaly-detection research demonstrates that industry-specific data and operational constraints can materially affect model behavior [41]. Therefore, maintenance agents should not be evaluated only on public Python repositories if the intended deployment includes banking, healthcare, insurance, CRM, or edge-cloud systems.

9. Discussion

AutoMaint-RAG reframes autonomous software maintenance as a systems-engineering problem. The central technical claim is that agentic patch generation becomes more reliable when it is constrained by retrieval grounding, localization evidence, executable feedback, and governance policy. The central organizational claim is that autonomous maintenance should augment, not bypass, existing software lifecycle controls.

The framework has direct relevance to enterprise systems where AI already supports lifecycle optimization, cybersecurity intelligence, and software-quality governance [34]. In such environments, the key value of agentic AI is not replacing developers but reducing the time spent on repetitive localization, reproduction, test selection, and patch drafting. Human engineers remain responsible for ambiguous requirements, high-risk architectural decisions, and final approval of sensitive changes.

The approach also aligns with research on ML-based anomaly detection, fraud detection, and monitoring because maintenance often begins with operational symptoms rather than clear bug reports [32]. For example, a production incident may appear as a latency anomaly, failed transaction, or data discrepancy. AutoMaint-RAG can ingest such signals into its retrieval layer, connect them to code paths, and propose candidate patches only when sufficient evidence exists.

The framework is particularly suitable for cloud-native and microservice systems. Work on HIPAA-compliant cloud architecture, prescription automation, and platform comparison shows that enterprise systems often combine strict compliance rules with heterogeneous deployment platforms [44]. Autonomous maintenance in these settings requires platform awareness, audit logging, and rollback planning. It cannot be safely handled by a model that only sees a source-code excerpt. Healthcare digital-transformation research further shows that clinical research and data-management technologies impose domain-specific validation and governance requirements [27].

There are also limits. First, agentic systems remain vulnerable to incomplete tests. If the available test oracle is weak, the agent may produce a plausible but incorrect patch. Second, retrieval can fail when code lacks meaningful names or when dependencies are implicit. Third, multi-agent systems can introduce coordination overhead and inconsistent intermediate assumptions. Fourth, governance policies may reduce autonomy in high-risk contexts, limiting fully automatic repair. These limitations do not undermine the framework; they define where human review and better evidence collection are necessary.

10. Threats to Validity

Construct validity depends on whether the selected metrics capture real maintenance quality. A patch that passes visible tests may still be semantically wrong. To mitigate this, the evaluation protocol includes regression breadth, mutation checks, generated-test review, and human inspection. Research on vulnerability detection using code metrics and feature extraction suggests that multiple evidence types are needed to assess software risk [72].

Internal validity depends on whether observed improvements come from the proposed architecture rather than from a stronger base language model. The evaluation must therefore fix the underlying model across baselines and vary only architecture components. Studies comparing neural-network optimization approaches show that algorithmic configuration can significantly influence measured performance [50]. This reinforces the need for controlled ablations.

External validity depends on repository diversity. A framework that works on Python libraries may not generalize to Java services, Salesforce platforms, cloud infrastructure, or data pipelines. Research on Salesforce CRM, DeFi intelligence, and healthcare engagement illustrates that enterprise platforms impose domain-specific constraints [65]. The evaluation should therefore include multiple languages, deployment patterns, and organizational policies.

Reliability depends on reproducibility. Each run should record model version, prompts, retrieval results, tool commands, test outputs, container images, and patch diffs. Monitoring and observability frameworks for cloud-based data pipelines demonstrate the importance of trace evidence for diagnosing AI-assisted operational decisions [73].

11. Conclusion

This paper proposed AutoMaint-RAG, a retrieval-augmented multi-agent framework for autonomous software maintenance. The framework decomposes maintenance into issue triage, retrieval, fault localization, patch planning, patch generation, test synthesis, regression validation, and governance. Its core contribution is the integration of agentic AI with software-engineering controls: evidence-grounded context selection, suspicious-location ranking, iterative execution feedback, regression-risk scoring, and auditable release decisions.

The paper argues that autonomous maintenance should not be understood as unrestricted code generation. Instead, it should be treated as a disciplined lifecycle process in which agents operate under explicit constraints and produce reviewable artifacts. This design is especially important for cloud-native, regulated, and enterprise systems where correctness, compliance, and rollback readiness are as important as patch generation.

Future work should implement the full framework across multiple benchmark families, measure cost and latency in CI/CD environments, and study human-agent collaboration during code review. Additional research should examine how agent memory can be safely reused across maintenance tasks without leaking sensitive code or reinforcing previous mistakes. Hybrid deep-learning fault-prediction research suggests that predictive signals can further strengthen the localization stage when combined with retrieval and execution feedback [70]. The long-term objective is not to remove engineers from maintenance but to create trustworthy autonomous collaborators that reduce debugging effort while preserving software reliability, security, and governance.

References

[1] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). SWE-bench: Can language models resolve real-world GitHub issues? arXiv:2310.06770.

[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] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. International Conference on Learning Representations.

[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] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems.

[6] 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

[7] Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). SWE-agent: Agent-computer interfaces enable automated software engineering. arXiv:2405.15793.

[8] 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

[9] Raikar, T. and Apelagunta, V. (2025). Implementing SAP Fiori in S/4HANA Transitions: Key Guidelines, Challenges, Strategic Implications, AI Integration Recommendations. Journal of Engineering Research and Sciences, 4(11), 1–9. https://doi.org/10.55708/js0411001

[10] 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

[11] 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

[12] Mansur, E., et al. (2025). RAGFix: Enhancing LLM Code Repair Using RAG and Dynamic Retrieval. arXiv preprint.

[13] 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

[14] Raikar, T. (2025). High-Performance In-Memory Computing: A Research Study on SAP S/4 HANA Database Layer. American Journal of Technology, 4(2), 93–113. https://doi.org/10.58425/ajt.v4i2.449

[15] 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

[16] Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. Advances in Neural Information Processing Systems.

[17] Mutyam, N. (2024). Graph-based modeling of service dependencies for predicting failure propagation in distributed systems. International Journal of Multidisciplinary Evolutionary Research, 5(1), 113–116. https://doi.org/10.54660/IJMER.2024.5.1.113-116

[18] 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

[19] Zhang, Y., Ruan, H., Fan, Z., & Roychoudhury, A. (2024). AutoCodeRover: Autonomous Program Improvement. arXiv: 2404.05427

[20] 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

[21] 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

[22] Ridnik, T., Kredo, D., & Friedman, I. (2024). Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering. arXiv:2401.08500

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

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

[25] Abreu, R., Zoeteweij, P., Golsteijn, R., & van Gemund, A. J. C. (2009). A practical evaluation of spectrum-based fault localization. Journal of Systems and Software, 82(11), 1780–1792.

[26] 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

[27] 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 Int. Conf. Recent Trends Comput. Sci. Technol. (ICRTCST), Jamshedpur, India, 2026, pp. 886–891, doi: 10.1109/ICRTCST68392.2026.11545210

[28] 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

[29] Badertdinov, I., Golubev, A., Nekrashevich, M., Shevtsov, A., Karasik, S., Andriushchenko, A., Trofimova, M., Litvintseva, D., & Yangel, B. (2025). SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents. arXiv: 2505.20411.

[30] 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

[31] 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

[32] 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

[33] Le Goues, C., Pradel, M., & Roychoudhury, A. (2019). Automated Program Repair. Communications of the ACM, 62(12), 56–65.

[34] Balerao, M. (2023). A converged artificial intelligence architecture for innovation, software lifecycle optimization, and cybersecurity risk mitigation. International Journal of Multidisciplinary Futuristic Development, 4(1), 117–120. https://doi.org/10.54660/IJMFD.2023.4.1.117-120

[35] 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

[36] Gunda SK, Yettapu SDR, Bodakunti S, Bikki SB. Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management, 2023 Mar. 30;4(1):102-8. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P112

[37] Tao, Y., Qin, Y., & Liu, Y. (2025). Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches. arXiv:2510.04905.

[38] 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

[39] 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

[40] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., & Clark, P. (2023). Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651.

[41] Bandari, S. S. G., Banda, S., & Naik, S. "Machine Learning (ML) based Anomaly Detection in Insurance Industries." Journal of Information Systems Engineering and Management 10(32s), 2026. https://doi.org/10.52783/jisem.v10i32s.5182

[42] 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

[43] Zhang, S., Ding, Y., Lian, S., Song, S., & Li, H. (2025). CodeRAG: Finding Relevant and Necessary Knowledge for Retrieval-Augmented Repository-Level Code Completion. arXiv:2509.16112

[44] 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

[45] 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

[46] 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

[47] Sivva, S. D. (2023). 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, 4(1), 600–604. https://doi.org/10.54660/.JFMR.2023.4.1.600-604

[48] Wang, J., He, Y., & Chen, H. (2024). RepoGenReflex: Enhancing Repository-Level Code Completion with Verbal Reinforcement and Retrieval-Augmented Generation. arXiv:2409.13122

[49] 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

[50] 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

[51] 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

[52] Xia, C. S., & Zhang, L. (2024). Keep the Conversation Going: Fixing 162 out of 337 Bugs for $0.42 Each Using ChatGPT. Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis.

[53] 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

[54] Gudi, S. R. (2025). Enhancing optical character recognition (OCR) accuracy in healthcare prescription processing using artificial neural networks. European Journal of Artificial Intelligence and Machine Learning, 4(6). https://doi.org/10.24018/ejai.2025.4.6.79

[55] 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

[56] Sai Krishna Gunda; An exploration of adaptive ensemble approaches in software fault detection: Balancing accuracy and robustness. AIP Conf. Proc. 7 January 2026; 3345 (1): 020211. https://doi.org/10.1063/5.0298093

[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] Basaveni Siri Mallika Rao, Sai Santosh Goud Bandari, "Replacing AI Agents for Backend," International Journal of Scientific Research in Engineering and Management. vol. https://doi.org/10.55041/IJSREM.NCFT011

[59] 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

[60] 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

[61] 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

[62] 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

[63] 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

[64] 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

[65] 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

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

[67] 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

[68] Gazzola, L., Micucci, D., & Mariani, L. (2019). Automatic Software Repair: A Survey. IEEE Transactions on Software Engineering, 45(1), 34–67.

[69] 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

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

[71] Srikanth Reddy Gudi. (2025). A Comparative Analysis of Pivotal Cloud Foundry and OpenShift Cloud Platforms. The American Journal of Applied Sciences, 7(07), 20–29. https://doi.org/10.37547/tajas/Volume07Issue07-03

[72] 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

[73] 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

[74] 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