ISSN: 2473-4901
v12 n6797

Trust but Verify: A Framework for Secure AI-Assisted Legacy Software Modernization

Autumn Sanders
ars6120@uncw.edu

Sydney Ross
ser9729@uncw.edu

Zealia Svecharnik
zs8464@uncw.edu

Kasey Miller
millerkc@uncw.edu

University of North Carolina Wilmington
Wilmington, NC, 28403, USA

Abstract

Artificial intelligence (AI) is increasingly being used to modernize legacy software systems and accelerate migration to cloud-native architectures. While AI-assisted code generation offers substantial improvements in development speed and productivity, it also introduces cybersecurity, governance, and software assurance risks that traditional secure development practices were not designed to address. This paper investigates how organizations can systematically modernize software and validate AI-generated code before deploying it to production cloud environments. Using a design-oriented conceptual framework grounded in the NIST AI Risk Management Framework (AI RMF), Secure Software Development Framework (SSDF), Zero Trust principles, and established software assurance practices. The primary contribution of this research is a repeatable framework that enables organizations to securely adopt AI-assisted legacy modernization while maintaining software quality, regulatory compliance, and operational resilience. The framework provides information systems leaders and Chief Information Officers (CIOs) with practical guidance for balancing the productivity benefits of AI with the security and governance requirements of mission-critical cloud modernization.

Keywords: Artificial Intelligence Governance, Legacy Modernization, AI-Generated Code, Software Assurance, Zero Trust, Secure Software Development

Trust but Verify: Evaluating the Security of AI-Generated Code in Legacy Modernization Projects

Autumn Sanders, Sydney Ross, Zealia Svecharnik, Kasey Miller

1. INTRODUCTION

Legacy software continues to support many mission-critical information systems across government, healthcare, transportation, finance, manufacturing, and other critical infrastructure sectors. Although these systems remain operationally essential, many are written in aging programming languages and depend on architectures that are increasingly difficult to maintain, secure, and integrate with modern cloud-native platforms. Traditional modernization efforts are often expensive, labor-intensive, and time-consuming, creating strong incentives for organizations to adopt artificial intelligence (AI) to accelerate code translation, refactoring, documentation, and migration activities.

Recent advances in large language models (LLMs) have demonstrated significant potential for automating software engineering tasks. Code-generation models such as StarCoder2, CodeLlama, and similar foundation models can assist developers by generating, translating, debugging, and documenting software at a substantially greater speed than traditional development methods. However, AI-generated code introduces new cybersecurity, governance, and software assurance challenges. Large language models may generate hallucinated dependencies, insecure coding practices, outdated libraries, software supply chain vulnerabilities, and semantic errors that are difficult to detect using conventional software testing techniques. Consequently, organizations cannot assume that AI-generated code is trustworthy simply because it compiles or executes successfully.

These challenges are particularly significant for organizations operating mission-critical and regulated information systems where software failures may affect operational resilience, regulatory compliance, or public safety. Existing secure software development frameworks provide valuable guidance for traditional software engineering but offer limited direction for validating AI-generated software artifacts throughout the modernization lifecycle. As organizations increasingly integrate AI into software development, new evaluation approaches are needed to combine software assurance, cybersecurity, governance, and operational resilience into a unified process.

This paper proposes a conceptual evaluation framework for secure AI-assisted legacy software modernization. For the purposes of this research, the research problem addresses legacy software, which refers to applications written in older programming languages (i.e., Ada, COBOL) that operate on architectures that are no longer actively supported by mainstream development and are increasingly difficult to integrate with cloud-native platforms. This definition intentionally excludes systems that can be migrated to in-house data centers but could be studied in future work. The proposed framework integrates AI guardrails, SHIM-layer isolation, characterization testing, Zero Trust principles, secure software development practices, and governance controls into a unified methodology for evaluating AI-generated code before production deployment. Unlike prior work that primarily focuses on AI code generation or cloud migration, this research emphasizes trustworthy AI adoption through continuous validation, software assurance, and governance. The resulting framework provides Chief Information Officers (CIOs), information systems leaders, and software engineering teams with a repeatable process for balancing the productivity benefits of AI-assisted modernization with the security and resilience requirements of mission-critical cloud environments.

2. RELATED LITERATURE

2.1. AI Governance and Software Assurance

Migrating legacy software to a cloud-native architecture introduces complexities, particularly when employing automated AI mechanisms. Agentic AI refers to artificial intelligence systems that can autonomously achieve specific goals with minimal human oversight (MIT Sloan, 2024). These systems

consist of AI agents that engage in decision-making to address problems in real time. It is important to understand that while large language models (LLMs) are reactive, responding to prompts with step-by-step instructions, Agentic AI systems are capable of carrying out multi-step tasks autonomously. The primary distinctions between LLMs and Agentic AI are summarized as follows in Table 1:

LLM vs Agentic AI: Key Differences		
Aspect	**LLM**	**Agentic AI**
Autonomy	Reactive	Proactive
Memory	Stateless (by default)	Stateful (persistent context)
Control Flow	Prompt driven	Goal-driven workflows
Tool Usage	Requires external integration	Built-in or orchestrated tool use
Task Scope	Single-step tasks	Multi-step, long-term goals
Feedback Adaptation	Limited	Dynamic, iterative feedback loops
Initiative	User-driven	Self-initiated

Table 1: LLM vs Agentic AI Key Differences

Implementing Agentic AI systems may heighten operational risks associated with legacy software, including the potential for privilege misuse, inadvertent system modifications, and increased susceptibility to adversarial manipulation. Therefore, it is essential to develop evaluation frameworks to oversee output for accuracy, safety, relevance, and task completion (Leanware, 2024).

Establishing guardrails can effectively mitigate hallucinations and misinformation, eliminate biased or harmful narratives, prevent abuse, protect privacy and data, address security vulnerabilities, and ensure compliance with regulatory guidelines (Xie et al., 2025). For older code where security is critical, it is crucial to integrate PII (Personally Identifiable Information) filters, tool safeguards, and rule-based protections. The incorporation of guardrails, safeguards, PII filters, and rule-based protections will enhance the agentic architecture, facilitate tool calling, support the Risk Assessment Framework (RAF), and improve feedback-guided generation. Dong et al. (2025) provide a comprehensive survey of LLM safeguarding mechanisms. They found that existing guardrail solutions such as Llama Guard and Nvidia NeMo represent only the simplest level of neural-symbolic designs, pointing to significant opportunities for more deeply integrated guardrail architecture. Here is an example of how this system would function: Deploying Agentic AI systems would increase operational risks in legacy software, including misuse of privileges, unintended system modifications, and exposure to adversarial manipulation. It is imperative to establish evaluation frameworks to monitor outputs for accuracy, safety, relevance, and task completion. Figure 1 illustrates how this system would work.

ISSN: 2473-4901
v12 n6797

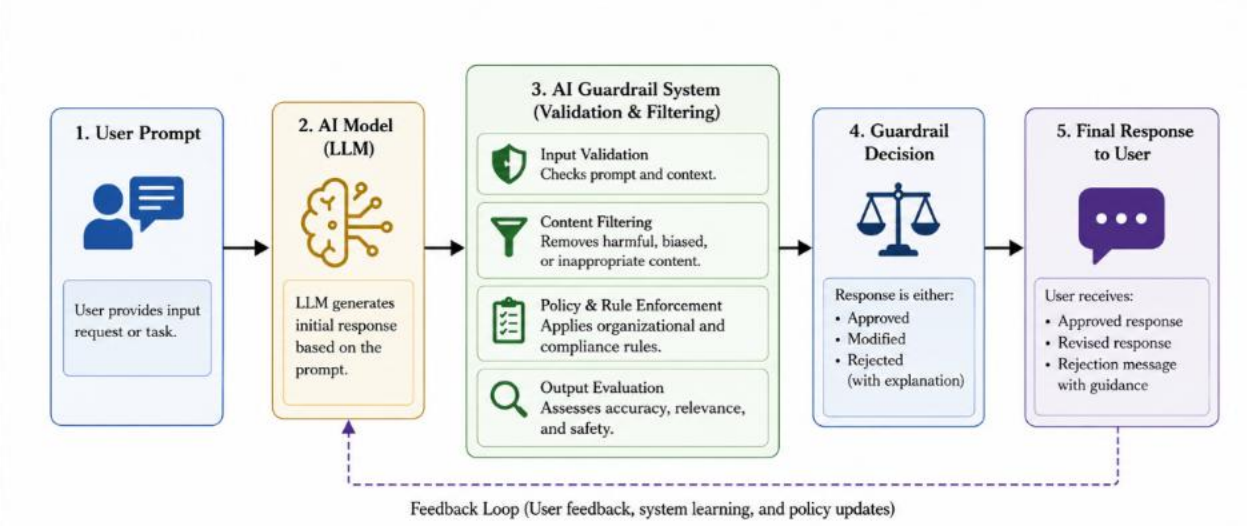

Figure 1. How the AI Guardrail System Response Works

While C++ consistently ranks among the top three programming languages (TIOBE, 2026), its widespread use in AI-assisted legacy software modernization creates the potential for disproportionate security risks in mission-critical systems when AI-generated code contains vulnerabilities. Zhang et al. (2024) evaluated ChatGPT-4 and Claude on 223 C/C++ code snippets spanning seven memory corruption vulnerability categories. They found that LLMs performed well on localized issues like memory leaks but struggled with complex vulnerabilities that required deeper reasoning. They further demonstrated that patch quality improved substantially when they provided the AI models additional context about variables, data structures, and code logic. A study published by the National Institute of Standards and Technology identified common risks in AI-generated C++ code, including memory mismanagement, buffer errors, and integer overflow (NIST, 2024). To mitigate these risks, C++ can be used to translate code into a more structured format, followed by human oversight of the AI-generator to ensure that proper control mechanisms, such as filters and auditing measures, are in place. An Agentic architecture is crucial in this context, as it allows for the delegation of tasks among various agents: one for generating responses, another for documenting code, a third for conducting transformations, and a reviewing agent dedicated to identifying any emergent vulnerabilities.

Among the available methodologies, the Ada-to-C++ transformation via feedback-guided generation stands out as particularly effective. This approach allows the model to generate code iteratively while receiving error feedback, enabling continuous refinement (Alrashedy, 2025). Additionally, leveraging external tools can significantly enhance this code transformation process. For instance, a tool-calling mechanism could be implemented, allowing a model to invoke external documentation retrieval systems to provide relevant resources for specific code segments. Additionally, integrating a Retrieval-Augmented Generation (RAG) model could facilitate the provision of multiple responses for document queries, thereby enriching the output (Kharma et al., 2026).

With AI Code-generation models (i.e., Starcoder2, LoRA, QLoRA), the underlying model and fine-tuning techniques such as LoRA (Low-Rank Adaptation) and QLoRA (Quantized Low-Rank Adaption) become essential, especially given the intricacies of transforming large codebases within sensitive frameworks. LoRA maintains the integrity of the base LLM while introducing a lightweight adaptive layer for training. Its memory efficiency, simplicity of implementation, low risk of overfitting, and parameter efficiency make it highly suitable for a wide range of tasks. Furthermore, LoRA adaptations can be seamlessly integrated into the AI code-generation model weights without incurring additional interference costs (GeeksforGeeks, 2024).

2.2. Shim Layer Architecture for Secure AI Modernization

A shim layer is a thin compatibility layer placed between legacy software and a modern environment so the old application can continue functioning without immediately rewriting all of its code. AI is set

to play a pivotal role in the modernization of legacy code by automating processes such as translating code from Ada to C++, refactoring, generating documentation, and analyzing logic. While the outputs of AI can often be probabilistic rather than deterministic, proper training based on the provided data, effective prompt structuring, and an understanding of inherent model limitations can enable AI to generate more reliable code.

2.3 AI-Assisted Code Transformation Risks

As AI facilitates the migration of legacy data to cloud-native environments, it may generate artifacts or dependencies that introduce vulnerabilities. Large Language Models (LLMs) often produce non-existent libraries or fabricate fictitious dependencies, and may reference outdated external components, thereby creating risks within the software supply chain (Li & Gao, 2025). Additionally, AI-generated outputs may include Common Vulnerabilities and Exposures (CVEs), insecure default configurations, or deprecated cryptographic methods, which heighten the potential for exploitation. LLMs can be trained on outdated information, potentially leading to the generation of fabricated external references and stale dependencies that attackers can exploit to hijack the software workflow or later exploit these dependencies. Zhang et al. (2024) identified four recurring failure modes in LLM generated patches: missing code context, introduction of new vulnerabilities, misidentification of the vulnerable sections, and misunderstanding of code logic. To mitigate such risks, it is essential to employ middleware tools that can detect suspicious code and fabricated components prior to integration, thereby reducing the likelihood of hallucinations and insecure references to components.

2.4 Cybersecurity Risks of AI-Generated Software

When utilizing AI to transform sensitive and regulated software, it is essential to treat all AI-generated outputs as untrusted inputs and to rigorously validate their security prior to integration into any production environment. It is crucial to acknowledge the potential risks of prompt injections, model manipulation, and unsafe automation. A thorough review of all inputs fed into the AI model is necessary to guarantee the safety of the output, to confirm the accuracy of the data received, and to ensure the AI produces dependable results. According to the OWASP Foundation (2025), secure design is critical for addressing these risks. Further, the use of external AI systems can pose risks of data leakage through inputs, outputs, logging, and retention. Consequently, establishing a contractual agreement with the vendor is vital to ensure that none of the data is disclosed.

AI-specific threats diverge from traditional application threats, as AI systems generate both content and logic. Characterization testing is beneficial and should be implemented during the initial stages of code transformation and again after the AI has been evaluated. Once testing has been conducted comprehensively, necessary adjustments to the content should be made. To identify flaws in newly generated code, an AI system that generates fuzzing harnesses for a Java library would be ideal (Loose et al., 2026). This concept can assist in creating harnesses that verify the translated code, uncover bugs and semantic mismatches, and ensure proper functionality, thereby avoiding potential misconfigurations.

2.5 AI Governance, Risk, and Compliance

According to the National Institute of Standards and Technology (NIST) Artificial Intelligence Risk Management Framework (AI RMF), effective AI risk management requires clearly defined accountability structures, organizational roles, responsibilities, and governance processes throughout the AI system lifecycle (NIST, 2023a). These requirements are particularly important when AI systems support security decision-making or software code generation. Organizations should establish clear ownership of AI systems and responsibility for their outputs while maintaining appropriate documentation, transparency, and human oversight. The use of third-party or vendor-managed AI services does not eliminate organizational responsibility for managing associated risks. NIST further recommends that organizations assess third-party generative AI providers for security, privacy, intellectual property, and legal compliance risks (NIST, 2023). Accordingly, auditability, explainability, and comprehensive documentation should be incorporated throughout the AI-enabled cloud environment.

Additionally, establishing contractual controls for AI vendors is vital for maintaining confidentiality, enforcing policy compliance, and ensuring human oversight of AI-generated outputs. This will help keep

operations aligned with intended outcomes. Ongoing evaluation of AI behavior is essential for effective long-term risk management as technology evolves. For AI code-generation models, we must also take into account the legal, risk, and compliance dimensions. Use of Starcoder2 is governed by the OpenRAIL-M license, which permits users to modify and fine-tune the model, provided it does not lead to legal violations, harm to individuals, or the generation of malware (Li et al., 2023). It is important to recognize that the output from the AI generation models may be inadequate, as it sometimes contains code with security vulnerabilities, unsafe memory usage, bugs, and erroneous logic.

Furthermore, AI code-generation models may reproduce portions of their training data, potentially introducing licensing and copyright concerns when generated code resembles existing source code (Li et al., 2025). These risks are compounded by the potential for AI-generated code to contain security vulnerabilities. Consequently, AI-generated code should undergo human review, security analysis, and testing before deployment, particularly in cloud and mission-critical environments (Booth et al., 2024).

2.6 Economic and Operational Considerations

The enhancements made by researchers with AI code generation models regarding parameter optimization, training methodologies, and dataset quality contribute to reduced training time, computational costs, and hardware requirements (Li et al., 2023). This facilitates a more efficient and cost-effective transition of code from Ada to C++. However, there is an inherent risk that the LLM will generate incorrect or insecure code, which may result in software malfunctions, security vulnerabilities, and system downtime—leading to financial losses and operational interruptions. In light of these potential challenges, it is essential to develop a comprehensive Business Continuity and Disaster Recovery (BCDR) plan to ensure the availability of alternatives in such scenarios.

Remaining gaps in the literature include a deeper understanding of risks, controls, and validation of migration from legacy code to novel AI-updated code. Therefore, our research addresses three questions:

RQ1. What cybersecurity risks are introduced when AI-generated code is used to modernize legacy software systems?

RQ2. Which software assurance controls most effectively reduce the risks associated with AI-generated code?

RQ3. How can organizations systematically validate AI-generated software prior to deployment into production cloud environments?

3. PROPOSED FRAMEWORK

3.1 Cloud Platform and Infrastructure Security (Post AI Deployment)

As AI modernization progresses, legacy code is transformed into modern code in cloud architecture, i.e., containers. Before deploying results to the cloud, it is essential to securely adapt AI artifacts to ensure a seamless transition from off-cloud to on-cloud. Since AI will function during the pre-deployment phase, it is imperative to ensure compliance with the critical infrastructure cloud computing requirements. Only validated outputs should be hosted within the cloud infrastructure. The integration of security boundary tools is critical to maintaining the AI within the confines of the cloud system architecture (NIST, n.d.).

After the cloud infrastructure has been deployed, it is crucial to re-establish the trust that the AI system may have lacked previously. Adhering to the AI RMF framework is vital to ensure that security controls are reapplied after code validation, confirming that no unauthorized adaptations have occurred (NIST, 2023). Implementing Identity and Access Management (IAM) access controls is also necessary to enforce the principle of least privilege for the deployed applications created by the AI-generated code and Cloud Infrastructure pipelines. Additionally, once the modernization code is finalized, it is important to remove AI credentials and concentrate on fortifying the network and infrastructure after deployment.

ISSN: 2473-4901
v12 n6797

3.2 The Shim Layer Architecture

An integral part of integrity assurance includes stress-testing the AI output. Additionally, it is strongly advised that an agentic model never executes code directly on cloud infrastructure. The proposed shim layer will intercept requests and format them according to the FIM (File-in-the-Middle) prompt template for AI code-generation models (e.g., StarCoder2). StarCoder2 is an open large language model (LLM) specifically trained for software code generation and completion. This approach will enable monitoring of the agentic AI's raw output. The shim layer will address the limitations of AI-code generation tool-calling capabilities by mapping protocols into the generated code blocks. Given the complexity of this shim layer, using micro-benchmarking to test would be ideal. A series of microbenchmarks will help effectively isolate and stress-test the shim interfaces. It is crucial to focus on testing both the memory and interface boundaries. While conducting these microbenchmarks, careful measurement of latency in relation to convenience trade-offs is essential (Hasan et al., 2020). Additionally, stress-testing the shim layer will be highly advantageous for assessing its potential workload capacity. Figure 2 illustrates how this shim layer would work within the system.

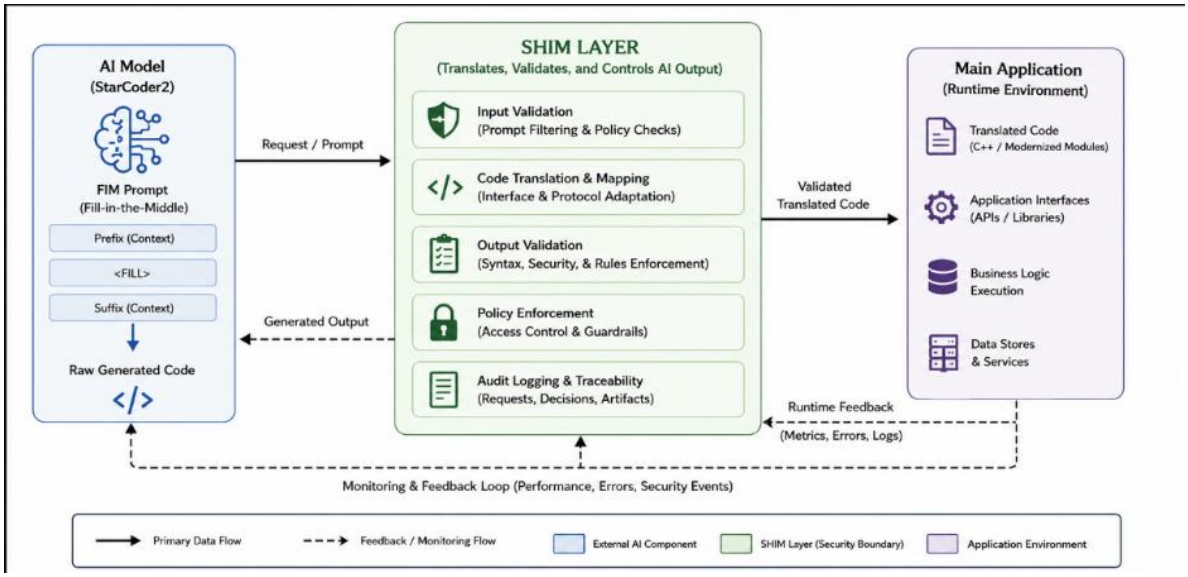

Figure 2: Visualization of SHIM layer translating code for main application

4. FRAMEWORK METHODOLOGY

4.1 Security Evaluation Criteria

This research adopts cybersecurity practices derived from established guidance, including the National Institute of Standards and Technology (NIST), Cybersecurity and Infrastructure Security Agency (CISA), and the United States Department of Defense Zero Trust Strategy. These practices provide a vendor-neutral framework applicable across critical infrastructure sectors and are integrated throughout the system development lifecycle to improve security and operational resilience.

The proposed framework evaluates and applies five core cybersecurity controls. These controls are: network segmentation, continuous monitoring, zero trust, resilience and recovery, and identity and configuration management. First, network segmentation limits lateral movement through logical separation of applications, services, and infrastructure using security groups, micro-segmentation, and policy enforcement. Second, continuous monitoring and threat intelligence integrate Security Information and Event Management (SIEM) platforms, threat feeds, Web Application Firewalls (WAFs), and DNS filtering to improve threat detection and response. Third, Zero Trust and secure software execution enforce identity verification, least-privilege access, digital code signing, and software integrity validation to reduce supply chain and insider risks. Fourth, resilience and disaster recovery incorporate automated backups, failover environments, and defined Recovery Time Objectives (RTOs) and Recovery Point Objectives (RPOs) to maintain operational continuity. Finally, identity and configuration management implements Role-Based Access Control (RBAC), secure configuration baselines, and

continuous monitoring of privileged accounts. The effectiveness of these controls is assessed through security validation techniques including penetration testing, configuration compliance analysis, vulnerability assessment, disaster recovery exercises, and privilege escalation testing. Table 2 below summarizes the five core capabilities evaluated by the framework, as well as the related standard and method of validation. Collectively, these activities provide a systematic methodology for evaluating the security and resilience of AI modernization in critical information systems.

Capability	Standard	Validation Technique
Network Segmentation	NIST SP 800-53, Zero Trust	Configuration compliance, pen test
Continuous Monitoring	CISA	SIEM review, WAFs, DNS filtering
Zero Trust	NIST AI RMF	Code signing, IAM assessment
Resilience + Recovery	RTO/RPO	Disaster recovery simulation, failover test
Identity + Configuration Management	IAM policy, NIST	RBAC audit, privilege testing

Table 2: The Five Core Cybersecurity Principles

5. FRAMEWORK VALIDATION

The proposed evaluation framework provides a structured methodology for assessing the security and operational readiness of AI-generated code during legacy system modernization. Rather than presenting empirical results, the framework synthesizes established software assurance, cybersecurity, and governance practices into a repeatable evaluation process for AI-assisted modernization. The framework integrates complementary validation activities that collectively assess functional correctness, software integrity, operational resilience, and infrastructure security before deployment into cloud-native environments (see Appendices A and B).

Characterization testing serves as the foundation of the framework by establishing behavioral equivalence between the original legacy application and the AI-generated implementation. Because large language models may preserve syntax while unintentionally altering program logic, characterization testing verifies that the translated application maintains the intended functionality across expected operating conditions. This approach provides confidence that modernization has not introduced semantic deviations before security validation begins.

Following functional verification, penetration testing evaluates the security posture of AI-generated code by identifying vulnerabilities that may arise during automated translation. These assessments consider traditional software weaknesses, including input validation, authentication, authorization, and privilege management, while also addressing AI-specific risks such as hallucinated dependencies, insecure code generation, and software supply chain exposure. The framework therefore recognizes that AI-assisted development introduces new attack surfaces that require validation beyond conventional secure coding reviews.

Signature validation extends software assurance by verifying the provenance and integrity of generated artifacts prior to deployment. By enforcing trusted certificates, signed executables, and approved software policies, the framework aligns with Zero Trust principles and reduces the likelihood of unauthorized or manipulated code entering production environments. This validation process strengthens software supply chain security while supporting regulatory and organizational compliance requirements.

The framework further evaluates operational resilience through disaster recovery simulations and failover testing. These activities assess the organization's ability to restore services following infrastructure failures while maintaining data integrity and system availability. Integrating resilience testing into the evaluation process acknowledges that secure modernization extends beyond vulnerability mitigation and includes maintaining mission continuity under adverse conditions.

Because the proposed architecture introduces an intermediary Shim layer between the AI model and production infrastructure, specialized performance evaluation is also incorporated. Micro-benchmark testing isolates individual Shim interfaces to examine processing overhead, protocol translation efficiency, and interface behavior, while stress testing evaluates scalability under increasing workloads. Together, these assessments determine whether the additional security controls can be implemented without introducing unacceptable performance degradation.

Collectively, these evaluation components form a layered assessment methodology that supports trustworthy AI-assisted modernization. Instead of viewing security as a final verification activity, the framework embeds validation throughout the modernization lifecycle, enabling organizations to evaluate AI-generated software from functional, security, operational, and governance perspectives before deployment.

6. DISCUSSION

The greatest risks associated with AI-assisted legacy modernization originate not from the large language model itself, but from the organizational tendency to trust AI-generated software without sufficient verification. The proposed framework provides a method to evaluate and assess these risks through the core 5 controls. This perspective reinforces the central premise of this research: AI should accelerate software modernization but should not replace secure software engineering practices.

The proposed framework also provides conceptual answers to the research questions. First (RQ1), AI-assisted legacy modernization introduces cybersecurity risks including hallucinated dependencies, insecure code generation, software supply chain vulnerabilities, prompt injection, and unauthorized system modifications. Second (RQ2), these risks can be mitigated through layered software assurance practices that combine characterization testing, penetration testing, secure software development controls, AI guardrails, and continuous governance rather than relying on any single security mechanism. Third (RQ3), organizations can establish trust in AI-generated software by implementing a structured validation process that verifies functional correctness, software integrity, infrastructure security, operational resilience, and governance compliance before deployment into production cloud environments

Characterization testing could prove particularly valuable because functional correctness represents the foundation upon which all subsequent security validation depends. Even when generated code successfully compiles, subtle behavioral differences may introduce logic flaws that traditional static analysis cannot easily detect. Ensuring semantic equivalence between the original Ada implementation and the translated C++ version reduces operational risk before security controls are evaluated.

Penetration testing results could further reinforce current guidance from NIST, OWASP, and the Department of Defense regarding AI-generated software. It’s critical to assess recurring concerns including hallucinated dependencies, insecure coding practices, outdated libraries, and insufficient input validation.

The shim layer seems the most promising architectural contribution of the proposed framework. Instead of allowing an external agentic AI model to interact directly with production infrastructure, the Shim layer creates an intermediary security boundary capable of validating prompts, translating interfaces, monitoring outputs, and enforcing organizational guardrails. Although the layer introduces modest processing overhead, the additional latency represents an acceptable trade-off considering the improvements in auditability, explainability, and security assurance.

Operational testing may also demonstrate that cybersecurity resilience extends beyond vulnerability prevention. Disaster recovery simulations and failover testing may show that organizations adopting AI-generated software must maintain the same expectations for availability, recovery, and continuity as traditionally developed applications. AI-generated code does not eliminate infrastructure failures, configuration errors, or operational disruptions; consequently, established business continuity and disaster recovery practices remain indispensable. Overall, the proposed evaluation framework provides

a defense-in-depth approach for securely adopting AI-assisted legacy modernization. By integrating AI guardrails, characterization testing, penetration testing, software integrity verification, operational resilience testing, and governance controls into a unified process, the framework provides organizations with a repeatable methodology for evaluating AI-generated software before deployment to production cloud environments.

7. CONCLUSION AND FUTURE WORK

This research advances the understanding of trustworthy AI adoption in legacy modernization by proposing a structured evaluation framework that fills existing gaps in software assurance, AI governance, and cloud security practices. This research argues that AI-generated code should be treated as an insecure artifact until it has undergone structured verification using secure software development, software assurance, and governance controls.

The primary contribution of this paper is the development of a conceptual evaluation framework for secure AI-assisted legacy modernization. Unlike existing approaches that primarily emphasize AI code generation or cloud migration, the proposed framework integrates AI guardrails, SHIM-layer isolation, characterization testing, Zero Trust principles, secure software development practices, and governance controls into a unified methodology for evaluating AI-generated code before production deployment. By embedding security validation throughout the modernization lifecycle, the framework provides organizations with a repeatable process for balancing the productivity advantages of AI with the security, compliance, and resilience requirements of mission-critical information systems. In the future, this framework can be empirically tested using Appendices A and B to modernize legacy software and move it to the cloud.

Although this research presents a conceptual framework, future work should empirically validate its effectiveness using real-world legacy modernization projects. Controlled case studies comparing AI-assisted and traditional modernization approaches could evaluate code quality, vulnerability density, software reliability, modernization effort, and deployment success. Additional research should investigate the effectiveness of SHIM-layer architectures, multi-agent validation workflows, automated software assurance techniques, and continuous governance mechanisms across different programming languages, cloud platforms, and critical infrastructure sectors. Such studies would provide quantitative evidence supporting the framework while refining best practices for trustworthy AI-assisted software modernization.

8. REFERENCES

Alrashedy, K., Aljasser, A., Tambwekar, P., & Gombolay, M. (2025). Leveraging Static Analysis for Feedback-Driven Security Patching in LLM-Generated Code. *Journal of Cybersecurity and Privacy*, *5*(4), 110. https://doi.org/10.3390/jcp5040110

Amazon Web Services. (n.d.). *What is a service mesh? - Service mesh explained*. https://aws.amazon.com/what-is/service-mesh

Booth, H., Souppaya, M., Vassilev, A., Ogata, M., Stanley, M., & Scarfone, K. (2024). Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile. NIST Special Publication 800-218A. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-218A

Dong, Y., Mu, R., Zhang, Y., Sun, S., Zhang, T., Wu, C., Jin, G., Qi, Y., Hu, J., Meng, J., Bensalem, S., Huang, X. (2025). Safeguarding large language models: a survey. *Aritifical Intelligence Review,* 58(382), 1-56. https://doi.org/10.1007/s10462-025-11389-2

GeeksforGeeks. (2024, February 21). *Fine-tuning using LoRA and QLoRA*. https://www.geeksforgeeks.org/fine-tuning-using-lora-and-qlora

Hasan, A., Riley, R., & Ponomarev, D. (2020). Port or shim? Stress testing application performance on Intel SGX. In 2020 IEEE International Symposium on Workload Characterization (IISWC) (pp. 166–177). IEEE. https://www.cs.cmu.edu/~rdriley/research/pubs/iiswc2020.pdf

Kharma, M., Sabbah, Ahmed., Alkhanafseh, M., Hammoudeh, M., & Mohaisen, D. (2026). An empirical evaluation of LLM-generated code security across prompting methods. *arXiv*. https://arxiv.org/abs/2605.24298

Leanware. (2024, December 18). *What Are Evals in AI? A Complete Guide to AI Evaluations.* https://leanware.co/insights/what-are-evals-in-ai

Li, X., & Gao, X. (2025). Investigating security implications of automatically generated code on the software supply chain. *arXiv*. https://doi.org/10.48550/arxiv.2509.20277

Li, R., Ben Allal, L., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., ... de Vries, H. (2023). StarCoder: may the source be with you! Transactions on Machine Learning Research. https://arxiv.org/abs/2305.06161

MIT Sloan. (2024, June 17). *Agentic AI, explained*. Ideas Made to Matter. https://mitsloan.mit.edu/ideas-made-to-matter/agentic-ai-explained

National Institute of Standards and Technology. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)* (NIST AI 100-1). U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.100-1

National Institute of Standards and Technology. (2023a). Artificial Intelligence Risk Management Framework: Generative AI Profile (NIST AI 600-1). U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.100-1

National Institute of Standards and Technology. (2024). Secure Software Development Framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST Special Publication 800-218). U.S. Department of Commerce. https://csrc.nist.gov/projects/ssdf

National Institute of Standards and Technology. (n.d.). System boundary. NIST Computer Security Resource Center Glossary. https://csrc.nist.gov/glossary/term/system_boundary

Office of Management and Budget. (2016, July). Circular No. A-130: Managing information as a strategic resource. Executive Office of the President. Obama White House Archives. https://obamawhitehouse.archives.gov/omb/circulars_a130

OWASP Foundation. (2025). OWASP top 10 for large language model applications. https://owasp.org/www-project-top-10-for-large-language-model-applications

Schwanke, A. (2023, December 21). Semantic layer — One layer to serve them all. Medium. https://medium.com/@axel.schwanke/semantic-layer-one-layer-to-serve-them-all-8a5c37e61e05

TIOBE. (2026). TIOBE index for June 2026. TIOBE Software. https://www.tiobe.com/tiobe-index/

Loose, N., Winkel, N., Hempel, K., Machtle, F., Hans, J., & Eisenbarth, T. (2026). Coverage-guided multi-agent harness generation for Java library fuzzing. Computers & Security, 161, Article 103850. https://www.sciencedirect.com/science/article/pii/S0167404826000180

Xie, M., Zhou, J., & Chen, H. (2025). AI guardrails in agentic systems explained: A systematic review of operational boundaries. Decision Support Systems, 189, Article 114320. https://www.sciencedirect.com/science/article/abs/pii/S0167923625001010

ISSN: 2473-4901
v12 n6797

Zhang, L., Zou, Q., Singhal, A., Sun, X., Liu, P. (2024) Evaluating Large Language Models for Real-World Vulnerability Repair in C/C++ Code. IWSPA 2024: Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics. https://doi.org/10.1145/3643651.3659892

ISSN: 2473-4901
v12 n6797

Appendices

APPENDIX A. AI-Assisted Legacy Modernization Evaluation Framework

To operationalize the proposed framework, this appendix presents evaluation criteria used to assess whether AI-generated code is suitable for deployment into production cloud environments. Rather than relying on a single security assessment, the framework applies multiple validation stages throughout the software modernization lifecycle.

Domain	Evaluation Objective	Validation Technique	Success Criteria
Functional Correctness	Preserve legacy system behavior	Characterization Testing	Functional equivalence with legacy application
Secure Coding	Identify software vulnerabilities	Static Analysis, Penetration Testing	No critical or high-risk vulnerabilities
Software Integrity	Verify provenance of generated code	Digital Signature Validation	All executables cryptographically verified
AI Output Validation	Detect hallucinated dependencies and insecure code	Manual Review + Dependency Analysis	No fabricated libraries or insecure dependencies
Identity & Access	Enforce least privilege	IAM/RBAC Assessment	Privilege escalation prevented
Operational Resilience	Maintain service continuity	Disaster Recovery & Failover Testing	RTO/RPO requirements satisfied
Infrastructure Security	Validate cloud deployment	Configuration Compliance Assessment	Compliance with security baseline
Governance	Maintain auditability	Documentation Review	Complete traceability of AI-generated artifacts

ISSN: 2473-4901
v12 n6797

APPENDIX B. Proposed Evaluation Metrics

Security Metrics
Critical CVEs
High CVEs
Hallucinated dependencies
Authentication vulnerabilities
Authorization vulnerabilities
Supply-chain vulnerabilities
Percentage of signed executables

Functional Metrics
Characterization test pass rate
Unit test pass rate
Integration test pass rate
Code coverage
Semantic equivalence score

Operational Metrics
Recovery Time Objective (RTO)
Recovery Point Objective (RPO)
Mean Time to Detect (MTTD)
Mean Time to Respond (MTTR)

Performance Metrics
SHIM layer latency
API response time
CPU utilization
Memory utilization
Throughput