AMERICAN ACADEMIC PUBLISHER

INTERNATIONAL JOURNAL OF DATA SCIENCE AND MACHINE LEARNING (ISSN: 2692-5141)

Volume 05, Issue 02, 2025, pages 412-418
Published Date: - 31-10-2025

Toward an Integrated AI-Driven Framework for Secure Code Transformation and Vulnerability Detection

Arjun Mehta
Department of Computer Science, GlobalTech University

ABSTRACT

The confluence of advances in machine learning, code translation, and automated security analysis presents transformative potential for modern software engineering. This article proposes a conceptual, integrated framework that leverages unsupervised code translation, large language models (LLMs) for code editing, and AI-enabled vulnerability detection, to enhance both code portability and security assurance. Drawing upon recent work in unsupervised programming language translation (Lachaux, Roziere, Chanussot & Lample, 2020), performance-improving code edits via learning (Shypula et al., 2023), zero-shot vulnerability repair with LLMs (Pearce et al., 2023), and broader surveys of AI-assisted big code understanding (Wong et al., 2023), the framework emphasizes an iterative pipeline: translate → adapt → secure. Additionally, we integrate insights from research on AI-based vulnerability detection in enterprise contexts (Rajapaksha et al., 2023; Behfar, 2023), semantic deduplication of security findings (Gulraiz, 202x), and compliance automation (Amaral et al., 2021; Areo, 2021), to support compliance and operational deployment. Through detailed theoretical elaboration, we examine potential benefits, challenges, limitations, and future directions. We argue that such a unified pipeline can significantly reduce the human burden in cross-language migration and bolster resilience against security vulnerabilities, while acknowledging risks such as over-reliance on AI, calibration, and regulatory compliance.

KEYWORDS

AI-assisted programming, code translation, vulnerability detection, software security, large language models, compliance automation

INTRODUCTION

Modern software development increasingly relies on large, heterogeneous codebases written in different programming languages. As organizations evolve, there is a frequent need to migrate codebases—for maintainability, performance, compliance, or deployment platform alignment. However, manual translation is laborious, error-prone, and costly. At the same time, the growing prevalence of software vulnerabilities and compliance requirements (e.g., data privacy laws like GDPR) demands rigorous security assurance integrated within development processes. This dual challenge—porting code across languages while preserving or enhancing security and compliance—poses a significant engineering and research problem.

Recent advances in machine learning offer promising avenues. In particular, unsupervised translation techniques for programming languages (Lachaux et al., 2020) demonstrate the feasibility of translating code between languages without aligned parallel corpora. Simultaneously, large language models (LLMs) have shown capability for automated code edits that improve performance or adapt code to new contexts (Shypula et al., 2023), and even repairing vulnerabilities in a zero-shot setting (Pearce et al., 2023). Augmented by surveys of big code understanding (Wong et al., 2023) and analyses into the calibration and correctness of code LLMs (Spiess et al., 2024), the foundation for an AI-driven translation–adaptation–security pipeline seems within reach.

However, existing work remains fragmented. Research on code translation seldom integrates vulnerability analysis; AI–based vulnerability detection tools often operate on raw code without language migration considerations; compliance automation systems focus on policy-level artifacts (e.g., privacy policies) rather than operational code security. There is a conspicuous gap: no unified framework systematically combines code translation, adaptation/editing, vulnerability detection/repair, and compliance support.

In this article, we propose such a framework: an integrated AI-driven pipeline that transitions code across languages, adapts it to new contexts, detects and repairs vulnerabilities, and ensures compliance through semantic analysis—thus streamlining cross-language migrations while maintaining security and regulatory standards. We detail theoretical underpinnings, potential implementation strategies, evaluate benefits and limitations, and outline future research directions. Our contribution is conceptual clarity and synthesis, offering a roadmap for bridging disparate advances into a cohesive methodology.

METHODOLOGY

Given the conceptual nature of this work, our methodology is based on theoretical synthesis and systematic mapping of existing literature. We conduct a structured literature review of the provided references, categorize their contributions into functional components, and design an architectural framework that integrates them. The steps are as follows:

1. Mapping Core Capabilities

○ Identify and analyze fundamental capabilities demonstrated in literature: unsupervised code translation (Lachaux et al.), learned code editing for performance (Shypula et al.), zero-shot vulnerability repair (Pearce et al.), big code NLG and understanding (Wong et al.), and calibration of code LLMs (Spiess et al.).

○ Evaluate the generalizability of these capabilities across languages, codebases, and security contexts.

2. Define Pipeline Phases

○ Based on the capabilities, delineate a multi-phase pipeline: Translation, Adaptation/Editing, Security Analysis & Repair, Compliance & Deduplication & Deployment.

○ For each phase, specify inputs, outputs, responsibilities, and interactions.

3. Integrate Compliance and Operational Context

○ Examine literature on AI-based vulnerability detection in enterprise environments (Rajapaksha et al., Behfar, Ricol), semantic deduplication of findings (Gulraiz), and compliance automation for privacy/policy (Amaral et al., Areo).

○ Determine how security findings and compliance constraints can be semantically annotated, deduplicated, and tied back to code artifacts to support governance and regulatory compliance.

4. Critical Risk Assessment

○ Using insights from security engineering studies (Hermann et al., 2025), and calibration studies of LLMs (Spiess et al.), perform a risk and limitation analysis: trustworthiness of automated edits, potential for introducing vulnerabilities, overfitting to training distributions, false positives/negatives in detection, compliance mismatches.

5. Articulate Future Research Agenda

○ Identify open research questions, including empirical validation, human–in-the-loop guardrails, interpretability, auditability, and scaling to industrial-size repositories.

Through this methodology we build a comprehensive conceptual framework without generating empirical data. Instead, we rely on theoretical elaboration grounded in peer literature.

Results

Through the literature mapping and synthesis, we derive the following key results:

● Feasibility of Unsupervised Language Translation for Code

The work by Lachaux et al. (2020) demonstrates that unsupervised machine translation (initially developed for natural language) can be adapted to programming languages. Their approach does not rely on existing parallel corpora of code translations but instead uses monolingual corpora and applies back-translation and noise-robust training techniques. This indicates that, in principle, codebases written in, say, Python can be translated into Java or other languages without prior human-written translation pairs. The result suggests that cross-language migration of nontrivial codebases is technically feasible using unsupervised learning.

● Automated Code Edits for Performance and Functional Adaptation

Shypula et al. (2023) demonstrate that models can learn from edits that improve performance and generalize these edits to new code contexts. Their approach effectively treats "edits" as transformations learned through machine learning, enabling automation of nontrivial refactoring or optimization. This capacity becomes crucial when translated code must not only compile but also meet performance, style, or idiomatic standards of the target language or ecosystem.

● Vulnerability Detection and Zero-Shot Repair Using Large Language Models

The zero-shot vulnerability repair demonstrated by Pearce et al. (2023) shows that LLMs trained on large code corpora can identify common vulnerability patterns and propose corrective edits without fine-tuning on labeled vulnerability-repair pairs. This result establishes that LLMs inherently encode some understanding of secure coding practices, and can be leveraged to repair or mitigate vulnerabilities in code—potentially even code that has been automatically translated or edited.

● Comprehensive Understanding of Big Code and Code Generation via Natural Language

The survey by Wong et al. (2023) reviews the state-of-the-art in AI-assisted programming, including code generation, summarization, translation, and understanding. It underscores the breadth of "big code" tasks now addressable by AI, and highlights both potential and limitations, such as the risk of overfitting, hallucinations, and correctness/calibration issues.

● Calibration and Correctness Concerns in Code LLMs

The empirical work by Spiess et al. (2024) provides evidence that while LLMs for code can generate syntactically valid outputs, their calibration—the probability that a generated snippet is correct or secure—is often inadequate. This raises caution: automation without human oversight may lead to erroneous or insecure code.

● Enterprise and Compliance Contexts for AI-based Vulnerability Detection Tools

Studies such as Rajapaksha et al. (2023) and Behfar (2023) argue for the deployment of AI-based vulnerability detection tools in enterprise software lifecycles. Ricol (2022) outlines foundational aspects of "AI for secure software development." Additionally, semantic deduplication of security findings (Gulraiz) presents a pathway to manage security debt and reduce noise from overlapping vulnerability reports. Compliance automation for privacy policies (Amaral et al., 2021) and healthcare IT (Areo, 2021) further illustrate precedent for automating compliance-related tasks, though typically at higher abstraction layers (policy documents rather than code).

● Engineering Security Features and Privacy Constraints

The study by Hermann et al. (2025) on the engineering of security features emphasizes that security must be engineered deliberately and systematically, not as an afterthought. Their exploratory work shows that integrating security early in the development process improves outcomes.

From these findings, we concretize an integrated pipeline:

1. Translation Phase: Starting from a source codebase in language A, apply unsupervised code translation (Lachaux et al., 2020) to produce a target-language version.

2. Adaptation/Editing Phase: Employ learned code editing models (Shypula et al., 2023) or prompt-engineered LLMs (as explored by Zhang et al., 2024) to refactor translated code for idiomatic style, performance, or ecosystem compatibility.

3. Security Analysis and Repair Phase: Use LLM-based vulnerability detection and zero-shot repair (Pearce et al., 2023), supplemented with static–analysis AI tools as in enterprise vulnerability detection efforts (Rajapaksha et al., 2023; Behfar, 2023).

4. Semantic Deduplication & Compliance Phase: Apply semantic analysis techniques to deduplicate security findings (Gulraiz) and integrate compliance checking routines (Amaral et al., 2021; Areo, 2021), mapping code artifacts to compliance requirements (e.g., privacy constraints).

5. Human-in-the-Loop Review & Deployment: Given calibration concerns (Spiess et al., 2024), include human review and audit stages before deployment; optionally integrate continuous monitoring and feedback loops, aligning with practices of engineering security features (Hermann et al., 2025).

DISCUSSION

The framework described above offers several theoretical and practical advantages, but also raises important challenges and limitations.

Advantages

● Efficiency and Cost Reduction: By automating translation, adaptation, and even security repair, organizations can drastically reduce the manual effort required to migrate codebases, refactor legacy code, or port projects to new languages or platforms. Especially for large, monolithic codebases, this could mean substantial time and resource savings.

● Security by Construction: Integrating vulnerability detection and repair immediately after translation and editing helps prevent the classical pitfall of “letting technical debt build up,” where translated code remains unchecked and introduces security holes. By combining automated static and LLM-based analyses with semantic deduplication and compliance checks, the pipeline promotes a security-first mindset.

● Compliance Readiness: With increasing regulatory pressure (privacy laws, data handling standards), tying code artifacts to compliance checks via semantic analysis (“this function handles personal data — ensure compliance”) enables better readiness and traceability.

● Scalability and Repeatability: Once established, the pipeline can be reused across multiple projects, languages, and teams; it answers a systemic need rather than a one-off migration.

Challenges and Limitations

● Calibration and Correctness of LLM Outputs: As shown by Spiess et al. (2024), current code-focused LLMs may

lack reliable self-assessment; they might produce code that looks correct but has subtle bugs or security flaws. Overreliance on such automation without robust human review can be hazardous.

● Semantic Drift and Idiomatic Mismatch: Unsupervised translation may yield syntactic correctness but fail to capture idiomatic patterns or best practices of the target language's ecosystem. Editing models may only partially bridge this gap; fully idiomatic, maintainable code may still require human refactoring.

● Vulnerability Coverage Limitations: LLM-based zero-shot repair relies on patterns seen during training; rare or novel vulnerabilities, complex logic flaws, or non-code-based vulnerabilities (e.g., design-level, architectural) may evade detection or repair. Static analysis tools may also have limited coverage or miss context-dependent issues.

● Compliance Complexity and Interpretability: Mapping code artifacts to regulatory compliance requirements (e.g., GDPR, HIPAA) is nontrivial. Semantic analysis might produce false positives/negatives; the behavior of transformed code may differ in deployment, invalidating compliance checks. Auditors may demand human-readable proofs or traceability, which AI-generated transformations may complicate.

● Human Trust and Adoption Barriers: Developers and security teams may resist fully automated pipelines due to distrust of "black-box" transformations. Integration into existing development workflows, continuous integration/continuous delivery (CI/CD), and organizational policies may pose friction.

● Maintenance and Update Challenges: As languages evolve, ecosystems change, or compliance regulations shift, the pipeline may need frequent retraining or reconfiguration. Ensuring long-term sustainability will require dedicated maintenance efforts.

Given these trade-offs, the proposed framework should not be viewed as a fully automated replacement for human-driven software engineering. Rather, it is a supportive augmentation — aiding developers and security engineers, reducing mundane effort, and highlighting potential issues early. It can serve best in a human-in-the-loop model where automated suggestions are reviewed, refined, and approved by experts.

Future Research Directions

To realize the potential of this integrated framework, several research directions should be prioritized:

1. Empirical Validation on Real-world Codebases

Conduct case studies where large legacy codebases undergo translation and adaptation using this pipeline. Measure metrics such as correctness (compilation, tests), performance, maintainability (code metrics, readability), and security (vulnerabilities identified before and after). This will provide concrete evidence for gains and pitfalls.

2. Human-in-the-Loop Workflow Design

Investigate how human reviewers (developers, security engineers, compliance officers) can best integrate with automated suggestions. Design UI/UX tools for presenting code edits, vulnerability findings, compliance issues, and collecting feedback. Study human acceptance, trust, and efficiency.

3. Improving Calibration and Interpretability of LLM Outputs

Research methods to provide confidence scores, provenance tracking (what data or patterns led to a suggestion), and human-readable explanations for edits or repairs. This will help address trust and auditability concerns.

4. Extending Vulnerability Detection Coverage

Enhance LLM training with curated datasets of known vulnerabilities (including rare and complex ones), integrate static analysis, dynamic analysis, and formal verification techniques, and explore hybrid methods to improve

detection and repair coverage.

5. Compliance Reasoning and Policy-aware Code Generation

Explore linking natural-language compliance policies (e.g., GDPR, HIPAA) with code semantics. Develop techniques for mapping policy requirements to code-level constraints, generating compliance checks, and verifying that code transformations preserve or enforce such constraints.

6. Governance, Ethical, and Regulatory Implications

Study organizational, legal, and ethical challenges of automated code transformation and repair. Questions include accountability ("who is responsible if AI-generated code introduces a vulnerability?"), auditability, transparency, and compliance with regulatory standards.

7. Sustainability and Maintenance of AI-assisted Pipelines

Research strategies for keeping the pipeline updated: retraining on evolving languages/ecosystems, monitoring drift, integrating continuous learning, and establishing version control of AI models.

CONCLUSION

The rapid advancement of AI-assisted programming presents an unprecedented opportunity to transform how software is developed, migrated, and secured. By synthesizing recent advances in unsupervised code translation, learned code editing, zero-shot vulnerability repair, and semantic compliance automation, we propose a unified, integrated pipeline for secure code transformation. This framework aims to reduce human effort, accelerate cross-language migration, and embed security and compliance as first-class citizens in the software lifecycle.

Nevertheless, realizing this vision requires cautious implementation, human oversight, and significant further research. Calibration issues, vulnerability coverage, compliance complexity, and human trust are nontrivial challenges. We envision the framework operating under a human-in-the-loop paradigm: AI-generated suggestions augmented — not replaced — by human expertise. As future work explores empirical validation, interpretability, governance, and policy integration, this approach may offer a practical and scalable path toward safer, more maintainable, and regulation-compliant software systems.

REFERENCES

1. Lachaux, M. A.; Roziere, B.; Chanussot, L.; Lample, G. Unsupervised translation of programming languages. arXiv, 2020, arXiv:2006.03511.
2. Shypula, A.; Madaan, A.; Zeng, Y.; Alon, U.; Gardner, J.; Hashemi, M.; Neubig, G.; Ranganathan, P.; Bastani, O.; Yazdanbakhsh, A. Learning performance-improving code edits. arXiv, 2023, arXiv:2302.07867.
3. Pearce, H.; Tan, B.; Ahmad, B.; Karri, R.; Dolan-Gavitt, B. Examining zero-shot vulnerability repair with large language models. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–25 May 2023.
4. Wong, M. F.; Guo, S.; Hang, C. N.; Ho, S. W.; Tan, C. W. Natural language generation and understanding of big code for AI-assisted programming: A review. Entropy, 2023, 25, 888.
5. Hermann, K.; Peldszus, S.; Steghöfer, J. P.; Berger, T. An exploratory study on the engineering of security features. In Proceedings of the International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, 27 April–3 May 2025.
6. Spiess, C.; Gros, D.; Pai, K. S.; Pradel, M.; Rabin, M. R. I.; Alipour, A.; Jha, S.; Devanbu, P.; Ahmed, T. Calibration

and correctness of language models for code. In Proceedings of the International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024.

7. Zhang, T.; Yu, Y.; Mao, X.; Wang, S.; Yang, K.; Lu, Y.; Zhang, Z.; Zhao, Y. Instruct or interact? Exploring and eliciting LLMs' capability in code snippet adaptation through prompt engineering. In Proceedings of the International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024.
8. Rajapaksha, S.; Senanayake, J.; Kalutarage, H.; Al-Kadri, M. O. Enhancing security assurance in software development: AI-based vulnerable code detection with static analysis. In European Symposium on Research in Computer Security, September 2023. Springer Nature Switzerland.
9. Behfar, S. K. Development strategy and management of AI-based vulnerability detection applications in an enterprise software environment. ECIS 2023 Research-in-Progress Papers.
10. Ricol, J. AI for secure software development: Identifying and fixing vulnerabilities with machine learning. 2022.
11. Gulraiz, A. Semantic analysis for deduplication of security findings in DevOps security tool reports. (date unspecified).
12. Samtani, S.; Abate, M.; Benjamin, V.; Li, W. Cybersecurity as an industry: A cyber threat intelligence perspective. In The Palgrave Handbook of International Cybercrime and Cyberdeviance, 2019, Palgrave Macmillan, Cham.
13. Sun, N.; Ding, M.; Jiang, J.; Xu, W.; Mo, X.; Tai, Y.; Zhang, J. Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives. IEEE Communications Surveys and Tutorials, 25(3), 1748–1774, 2023.
14. Zhou, Y.; Tang, Y.; Yi, M.; Xi, C.; Lu, H. CTI view: APT threat intelligence analysis system. Security and Communication Networks, 2022(1), 9875199.
15. Areo, G. Automating compliance in healthcare IT: Essential tools and techniques. 2021.
16. Amaral, O.; Abualhaija, S.; Torre, D.; Sabetzadeh, M.; Briand, L. C. AI-enabled automation for completeness checking of privacy policies. IEEE Transactions on Software Engineering, 48(11), 4647–4674, 2021.
17. Tang, F. I. P. Making AI GDPR compliant. 2019.
18. Security and privacy testing automation for LLM-enhanced applications in mobile devices. International Journal of Networks and Security, 5(02), 30–41, 2025.