Frontiers in Emerging Engineering & Technologies (Open Access)

Volume 03, Issue 03, March 2026,
Publish Date: 01-03-2026
Page No.01-09

A Human-Centric Framework For Secure AI-Assisted Code Generation: Embedding Security Heuristics In The Software Synthesis Process

Dr. Kavindu Perera

Department of Emerging Engineering & Technologies
National Institute of Technology, Sri Lanka

Ms. Dinithi Fernando

Department of Emerging Engineering & Technologies
Research and Innovation Center, Sri Lanka

ABSTRACT

The rapid advancement of artificial intelligence (AI) has transformed software development by enabling automated code generation through large language models (LLMs). Contemporary AI coding assistants can generate functional software components, accelerate development cycles, and improve programmer productivity. However, growing evidence indicates that AI-generated code frequently contains security vulnerabilities, insecure coding patterns, and implementation flaws that may expose software systems to cyber threats. Existing studies demonstrate that AI-generated code often reproduces insecure practices learned from training data and may fail to adhere to established security standards. Consequently, organizations increasingly face challenges in balancing productivity gains from AI-assisted programming with software security requirements. This study proposes a human-centric framework for secure AI-assisted code generation that embeds security heuristics throughout the software synthesis process. The framework integrates human-in-the-loop validation, security-aware prompt engineering, vulnerability assessment, explainability mechanisms, and iterative feedback loops to improve code quality and security outcomes. A comprehensive review of literature on AI code generation, human-centered AI, software engineering, explainable machine learning, and cybersecurity is conducted to establish the theoretical foundation of the framework. The proposed model emphasizes collaborative intelligence between human developers and AI systems rather than complete automation. Findings indicate that integrating security heuristics with human oversight significantly enhances vulnerability detection, code reliability, and user trust while reducing the likelihood of insecure software deployment. The study contributes a structured conceptual model that supports secure AI adoption in software engineering and identifies future research directions for adaptive, explainable, and security-aware code synthesis systems.

KEYWORDS: AI-assisted programming, secure code generation, human-in-the-loop learning, software security, security heuristics, large language models, explainable AI, vulnerability assessment, software synthesis, human-centered AI

INTRODUCTION

1.1 Background

Artificial intelligence has emerged as one of the most transformative technologies in modern software engineering. Large language models trained on massive repositories of source code have enabled AI systems to generate software components, automate programming tasks, and assist developers in real-time coding environments. Recent advancements in transformer-based architectures have significantly enhanced machine capabilities in code understanding and generation, enabling models to produce syntactically correct and functionally relevant code across multiple programming languages (Vaswani et al., 2017).

The increasing adoption of AI coding assistants has introduced substantial efficiency improvements within software development workflows. Studies evaluating large

language models trained on code demonstrate remarkable performance in code synthesis, code completion, and problem-solving tasks (Chen et al., 2021). Similarly, assessments based on benchmark datasets reveal that advanced AI systems can solve complex programming challenges with increasing accuracy (Hendrycks et al., 2021). These developments suggest that AI-assisted programming may become a foundational component of future software engineering practices.

Despite these advancements, significant concerns remain regarding the security implications of AI-generated code. Security analyses of AI coding assistants indicate that generated code often includes vulnerabilities, insecure authentication mechanisms, improper input validation, and unsafe cryptographic implementations (Pearce et al., 2022). Empirical investigations further reveal the presence of security smells and insecure programming constructs within AI-generated software artifacts (Siddiq & Bissyande, 2023). Such findings raise important questions regarding the reliability of autonomous code generation systems in security-sensitive environments.

The challenge is particularly relevant because software vulnerabilities remain among the primary causes of cyberattacks. The OWASP Top Ten framework identifies recurring categories of application security risks, while MITRE's CWE Top 25 catalog highlights critical software weaknesses commonly exploited by attackers (OWASP, 2021; MITRE, 2023). If AI systems reproduce these weaknesses during code generation, the resulting software may inherit significant security risks.

Human-centered artificial intelligence offers a promising approach to addressing these challenges. Human-in-the-loop methodologies emphasize collaboration between automated systems and human expertise, ensuring that critical decisions remain subject to human oversight (Monarch, 2021). Rather than replacing software developers, AI systems can function as intelligent assistants whose outputs are continuously evaluated, refined, and validated by human experts.

A particularly relevant insight emerges from explainable learning research, where model outputs become more trustworthy when explanations align with human reasoning processes (Ross et al., 2017). Similarly, visualization and interpretability techniques contribute to understanding internal model behavior and improving user confidence in AI-generated outcomes (Karpathy, Johnson, & Fei-Fei, 2015). These principles can be extended to secure code generation environments where transparency and explainability support security validation activities.

1.2 Problem Statement

Although AI-assisted code generation improves development productivity, existing systems frequently generate code containing security vulnerabilities. Current approaches emphasize functional correctness and programming efficiency but often neglect systematic security evaluation during code synthesis. The absence of integrated security heuristics and structured human oversight creates risks associated with deploying insecure software generated by AI systems.

1.3 Research Objectives

The objectives of this study are:

1. To analyze existing research on AI-assisted code generation and software security.
2. To identify security challenges associated with AI-generated code.
3. To develop a human-centric framework that embeds security heuristics into AI-assisted software synthesis.
4. To examine the role of explainability and human oversight in improving secure code generation.
5. To evaluate the theoretical implications of integrating security validation mechanisms within AI coding workflows.

1.4 Scope and Significance

This study focuses on AI-assisted code generation systems, human-in-the-loop methodologies, software security frameworks, and explainable artificial intelligence. The proposed framework contributes to secure software engineering by integrating security-focused decision-making throughout the AI-assisted development lifecycle. The research is significant because it addresses the growing need for trustworthy AI programming systems capable of producing secure and reliable software artifacts.

2. Literature Review

2.1 Evolution of AI-Assisted Code Generation

The emergence of transformer-based architectures fundamentally changed machine learning applications in software development. The attention mechanism introduced by transformer models enabled efficient processing of sequential information and facilitated substantial improvements in natural language understanding and code generation capabilities (Vaswani et al., 2017). These architectural advancements established the foundation for modern AI coding assistants.

Research evaluating large language models trained on source code demonstrated that AI systems can generate meaningful programming solutions across diverse software engineering tasks (Chen et al., 2021). Such models learn programming patterns from large-scale repositories and subsequently apply these patterns to generate code based on user prompts. Performance evaluations indicate significant gains in productivity and coding efficiency.

The APPS benchmark introduced by Hendrycks et al. (2021) further highlighted the growing competence of AI systems in solving programming challenges. Results suggested that AI-generated solutions increasingly approximate human-level programming performance across a range of coding tasks.

However, code generation accuracy does not necessarily imply software security. While generated outputs may satisfy functional requirements, they may simultaneously contain hidden vulnerabilities that become exploitable in production environments.

2.2 Security Challenges in AI-Generated Code

Security has emerged as a major concern within AI-assisted programming environments. Pearce et al. (2022) conducted one of the earliest systematic evaluations of AI-generated code security and found that coding assistants frequently produce insecure implementations. Vulnerabilities appeared particularly prevalent in authentication, cryptography, and input validation scenarios.

Further empirical analysis by Siddiq and Bissyande (2023) identified multiple categories of security smells in AI-generated code. These findings suggest that language models may inadvertently reproduce insecure programming practices observed in training datasets. Consequently, generated code may propagate historical vulnerabilities into future software systems.

Sobania et al. (2023) extended this discussion by examining both security and code quality dimensions of AI-generated software. Their findings indicated that code quality improvements do not automatically translate into improved security outcomes. The distinction between functionality and security remains critical when evaluating AI-generated software artifacts.

The prevalence of software vulnerabilities documented within OWASP Top Ten and CWE Top 25 frameworks further underscores the importance of security-aware code generation mechanisms (OWASP, 2021; MITRE, 2023). Without systematic safeguards, AI-generated code may unintentionally reproduce many of these known weaknesses.

2.3 Human-Centered AI and Human-in-the-Loop Learning

Human-centered AI emphasizes collaboration between humans and intelligent systems rather than complete automation. Monarch (2021) argues that human expertise remains essential for decision validation, annotation, monitoring, and quality assurance processes within machine learning systems.

Human-in-the-loop paradigms are particularly relevant in software security contexts because vulnerability assessment often requires contextual understanding beyond purely statistical prediction mechanisms. Human developers possess domain knowledge, architectural awareness, and security expertise that AI systems may lack.

The future of software engineering increasingly involves collaborative workflows in which human developers supervise intelligent programming assistants rather than manually implementing every software component (Ernst, 2017). Such collaboration enables organizations to leverage automation benefits while maintaining accountability and quality control.

2.4 Explainability and Trust in AI Systems

Trust remains a critical factor influencing adoption of AI-generated software. Explainability research suggests that users are more likely to trust AI outputs when they understand the reasoning processes underlying model decisions (Ross et al., 2017).

Visualization techniques contribute significantly to interpretability by exposing internal representation patterns and decision pathways within neural networks (Karpathy, Johnson, & Fei-Fei, 2015). These insights support transparency and facilitate more informed human evaluation of machine-generated outputs.

In secure code generation environments, explainability mechanisms may provide developers with security rationales, vulnerability indicators, and confidence estimates. Such capabilities can improve detection of risky implementations and enhance overall trustworthiness.

Notably, the principles discussed by Karpathy, Johnson, and Fei-Fei (2015) demonstrate how understanding internal model behavior can reveal hidden patterns influencing generated outputs. Applying similar interpretability concepts to AI coding assistants may improve vulnerability awareness and support secure software development decisions.

2.5 Research Gap

Existing studies primarily focus on evaluating security vulnerabilities within AI-generated code rather than designing integrated frameworks that proactively mitigate security risks during code synthesis. While human-in-the-loop learning, explainability, and software security have been investigated independently, limited research has examined their integration within a unified secure code generation framework.

Furthermore, existing literature provides insufficient guidance regarding how security heuristics should be embedded throughout the AI-assisted development lifecycle. This gap motivates the development of a comprehensive human-centric framework that combines AI generation capabilities with security-aware human oversight, explainability mechanisms, and vulnerability assessment processes.

3. Methodology

3.1 Research Design

This study adopts a conceptual research and review methodology aimed at synthesizing knowledge from software engineering, cybersecurity, explainable artificial intelligence, and human-centered machine learning. Rather than evaluating a specific software tool, the research develops a theoretically grounded framework based on insights derived from the reviewed literature.

The methodology follows four sequential stages:

1. Literature synthesis.
2. Security challenge identification.
3. Framework construction.
4. Theoretical validation through comparative analysis.

These stages collectively support the development of a human-centric secure code generation model capable of addressing vulnerabilities associated with AI-assisted programming.

3.2 Theoretical Foundation of the Framework

The proposed framework is grounded in four complementary theoretical perspectives:

Human-in-the-Loop Learning

Human oversight serves as the primary control mechanism for validating AI-generated code. Human reviewers evaluate security implications, confirm compliance requirements, and provide corrective feedback to the AI system (Monarch, 2021).

Explainable Artificial Intelligence

Interpretability mechanisms enable developers to understand why specific code segments were generated. Explainability improves trust, supports auditing activities, and facilitates vulnerability identification (Ross et al., 2017).

Security Heuristics

Security heuristics represent practical rules derived from OWASP Top Ten and CWE Top 25 vulnerability classifications. These heuristics guide both AI generation and human evaluation processes (OWASP, 2021; MITRE, 2023).

Collaborative Intelligence

The framework assumes that optimal security outcomes emerge from collaboration between human expertise and machine intelligence rather than complete reliance on either component alone. This perspective aligns with contemporary visions of future software engineering (Ernst, 2017).

3. Methodology

3.3 Proposed Human-Centric Secure Code Generation Framework

The proposed framework consists of six interconnected layers designed to integrate security evaluation into every stage of AI-assisted software synthesis.

Layer 1: Requirement and Security Context Definition

The process begins with the specification of functional requirements and security expectations. Developers define software objectives, target environments, compliance requirements, authentication needs, privacy constraints, and threat considerations before interacting with the AI system.

Traditional AI code generators often receive incomplete prompts focused primarily on functionality. The proposed framework addresses this limitation by requiring explicit security context definition. For example, if an application processes user credentials, security requirements related to

password hashing, encryption, session management, and access control must be specified before code generation begins.

Embedding security expectations at the requirement stage reduces ambiguity and encourages generation of security-aware software components.

Layer 2: Security-Aware Prompt Engineering

Prompt engineering functions as a critical interface between human developers and AI systems. Security-aware prompts explicitly instruct the model to follow secure programming practices, avoid known vulnerabilities, and comply with established security standards.

For example, instead of requesting:

"Generate a login system."

The framework recommends:

"Generate a secure login system implementing password hashing, input validation, protection against SQL injection, secure session management, and compliance with OWASP security recommendations."

This stage effectively embeds security heuristics before code generation occurs.

Research evaluating AI-generated code indicates that output quality strongly depends on input specifications (Chen et al., 2021). Therefore, prompt engineering becomes a security control mechanism rather than merely a usability feature.

Layer 3: AI-Based Code Synthesis

At this stage, the language model generates source code based on user requirements and security-aware prompts.

Modern transformer architectures enable contextual understanding of programming instructions and software patterns (Vaswani et al., 2017). However, because AI models learn from historical code repositories, generated outputs may contain inherited vulnerabilities or insecure programming practices (Pearce et al., 2022).

Consequently, generated code should be treated as a preliminary artifact requiring systematic evaluation rather than a finalized software product.

The framework therefore rejects fully autonomous deployment and instead positions AI-generated code as a candidate solution subject to subsequent verification stages.

Layer 4: Automated Security Heuristic Assessment

The fourth layer introduces automated security validation mechanisms based on established vulnerability frameworks.

The assessment module evaluates generated code against:

- OWASP Top Ten risk categories.
- CWE Top 25 software weaknesses.
- Common authentication flaws.
- Input validation vulnerabilities.
- Cryptographic implementation errors.
- Access control weaknesses.
- Injection attack opportunities.

Each generated component receives a security risk score based on heuristic analysis.

For example, detection mechanisms may identify:

- Hardcoded credentials.
- Unsafe database queries.
- Weak encryption algorithms.
- Missing input sanitization.
- Excessive permissions.

This automated review provides an initial security filter before human evaluation.

Layer 5: Explainability and Transparency Layer

One of the most significant limitations of AI-generated software is the opacity of model decision-making processes.

To address this challenge, the framework incorporates an explainability layer inspired by interpretability research (Karpathy, Johnson, & Fei-Fei, 2015).

The explainability component provides:

- Rationale for generated code structures.
- Confidence indicators.
- Security-sensitive code highlights.
- Dependency explanations.
- Vulnerability risk summaries.

Visualization and interpretability mechanisms improve human understanding of generated outputs and facilitate informed decision-making.

The importance of transparency is further supported by research demonstrating that interpretable AI systems are more likely to gain user trust and support effective oversight (Ross et al., 2017).

Notably, concepts introduced by Karpathy, Johnson, and Fei-Fei (2015) illustrate how model visualization can reveal internal patterns influencing generated outputs. Similar approaches can enhance secure code auditing by exposing generation pathways associated with potential vulnerabilities.

Layer 6: Human Security Review and Feedback Loop

The final layer introduces human oversight.

Security analysts, software architects, and experienced developers evaluate generated code based on:

- Security compliance.
- Functional correctness.
- Architectural consistency.
- Organizational policies.
- Regulatory requirements.

Human reviewers identify contextual risks that automated tools may overlook.

Examples include:

- Business logic vulnerabilities.
- Domain-specific threats.
- Regulatory non-compliance.
- Architectural weaknesses.

Corrective feedback is subsequently incorporated into future generation cycles, creating a continuous improvement mechanism.

The iterative nature of this process aligns with human-in-the-loop learning principles (Monarch, 2021).

3.4 Security Heuristic Integration Model

The framework embeds security heuristics across five dimensions:

Preventive Heuristics

Preventive heuristics aim to avoid vulnerability creation during code synthesis.

Examples include:

- Secure defaults.
- Principle of least privilege.
- Input validation requirements.
- Safe API usage.

Detective Heuristics

Detective mechanisms identify vulnerabilities after code generation.

Examples include:

- Static analysis.
- Vulnerability scanning.
- Dependency inspection.

Corrective Heuristics

Corrective heuristics support remediation of identified security weaknesses.

Examples include:

- Secure code replacement.
- Automated patch suggestions.
- Refactoring recommendations.

Explainability Heuristics

Explainability heuristics provide transparency regarding security decisions.

Examples include:

- Risk annotations.
- Security reasoning summaries.
- Confidence indicators.

Human Validation Heuristics

Human validation ensures contextual and organizational suitability.

Examples include:

- Expert review.
- Compliance verification.
- Architectural assessment.

3.5 Framework Workflow

The workflow can be summarized as follows:

Security Requirements → Security-Aware Prompt → AI Code Generation → Automated Security Assessment → Explainability Layer → Human Review → Feedback Integration → Secure Deployment

Unlike traditional AI coding systems, the proposed workflow embeds security validation before deployment rather than after vulnerability discovery.

4. Results / Findings

The analysis suggests that secure AI-assisted code generation requires a transition from automation-centric approaches toward collaborative human-AI workflows. Literature consistently demonstrates that AI-generated code can improve programming efficiency while simultaneously introducing security vulnerabilities (Pearce et al., 2022; Siddiq & Bissyande, 2023; Sobania et al., 2023).

The proposed framework addresses this challenge through layered security controls integrated across the software synthesis lifecycle. Findings indicate that security-aware prompt engineering can reduce ambiguity and guide generation toward safer implementations. Automated heuristic assessment provides scalable vulnerability screening, while explainability mechanisms enhance transparency and trust.

Human oversight emerges as the most influential component of the framework. Human reviewers possess contextual knowledge unavailable to machine learning models and can identify risks that escape automated detection. This observation supports human-centered AI principles emphasizing collaboration rather than replacement (Monarch, 2021).

The integration of explainability mechanisms further strengthens framework effectiveness. Research on neural network visualization suggests that understanding model behavior improves confidence in AI outputs (Karpathy, Johnson, & Fei-Fei, 2015). Applying these principles to code generation enables developers to evaluate security-sensitive decisions more effectively.

The framework also demonstrates strong alignment with established vulnerability taxonomies such as OWASP Top Ten and CWE Top 25. Embedding these standards directly into generation and validation processes increases the likelihood of producing secure software artifacts.

Overall, the findings indicate that combining AI-assisted programming with security heuristics, explainability, and human review can substantially improve software security outcomes while preserving productivity benefits.

5. Discussion

The findings contribute to ongoing discussions regarding the role of artificial intelligence in software engineering. Existing research has largely focused on measuring coding performance and identifying vulnerabilities within generated code. However, less attention has been devoted to designing integrated governance mechanisms capable of mitigating these risks during software synthesis.

The proposed framework extends previous studies by introducing security as a continuous process rather than a post-development activity. This perspective aligns with secure software engineering principles and reflects the growing need for proactive cybersecurity practices.

One significant implication involves the relationship between productivity and security. AI coding assistants can accelerate development, but increased speed may inadvertently amplify vulnerability propagation if adequate safeguards are absent. The framework addresses this tension by embedding security checkpoints throughout the generation lifecycle.

The integration of explainability mechanisms represents another important contribution. Previous interpretability research demonstrated the value of understanding neural network behavior (Karpathy, Johnson, & Fei-Fei, 2015). Within secure code generation environments, explainability enables developers to assess security risks, understand generation rationale, and identify suspicious implementation patterns. Consequently, transparency becomes a critical component of trustworthy AI-assisted programming.

Comparison with existing literature reveals strong consistency regarding the need for human oversight. Security analyses of AI-generated code repeatedly emphasize limitations in autonomous vulnerability detection (Pearce et al., 2022; Siddiq & Bissyande, 2023). Human reviewers therefore remain indispensable participants in secure development workflows.

Despite its strengths, the framework possesses limitations. First, the study is conceptual and does not include empirical implementation or experimental validation. Second, security heuristics may require adaptation across programming languages, application domains, and organizational environments. Third, explainability mechanisms for large language models remain an evolving research area with unresolved challenges related to interpretability accuracy and scalability.

Future research should focus on implementing prototype systems based on the proposed framework and evaluating their effectiveness across real-world development environments. Comparative experiments involving traditional AI coding assistants and security-enhanced human-centric systems would provide valuable evidence regarding framework performance.

6. Conclusion

Artificial intelligence is rapidly reshaping software engineering by enabling automated code generation and intelligent programming assistance. While these technologies offer substantial productivity benefits, growing evidence indicates that AI-generated code may contain security vulnerabilities capable of undermining software reliability and exposing systems to cyber threats. The increasing integration of AI into software development therefore necessitates frameworks that balance automation efficiency with rigorous security assurance.

This study examined the security implications of AI-assisted code generation and proposed a human-centric framework that embeds security heuristics throughout the software synthesis lifecycle. Drawing upon literature in AI code generation, cybersecurity, explainable artificial intelligence, human-in-the-loop learning, and software engineering, the framework was designed to address vulnerabilities commonly associated with autonomous code generation systems.

The proposed framework incorporates six interconnected components: security context definition, security-aware prompt engineering, AI-based code synthesis, automated security heuristic assessment, explainability mechanisms, and human security review. Collectively, these components establish a layered defense strategy that reduces vulnerability propagation while preserving the productivity advantages of AI-assisted programming.

The analysis demonstrates that human oversight remains a critical requirement for secure software development. Although modern large language models exhibit impressive coding capabilities, they continue to inherit limitations associated with training data quality, contextual reasoning, and security awareness. Human reviewers provide contextual understanding, architectural judgment, and domain expertise that remain difficult to replicate through automated systems alone. Consequently, collaborative intelligence emerges as a more reliable paradigm than complete automation.

Another important contribution of this research is the integration of explainability into secure code generation workflows. Interpretability mechanisms support transparency, facilitate vulnerability assessment, and improve user trust in AI-generated outputs. Consistent with prior work on neural network visualization and understanding, explainability enhances developers' ability to evaluate and validate machine-generated decisions (Karpathy, Johnson, & Fei-Fei, 2015). By making code generation processes more transparent, organizations can establish stronger governance and accountability mechanisms.

The framework also demonstrates alignment with recognized cybersecurity standards such as the OWASP Top Ten and CWE Top 25. Embedding these security principles directly into generation and evaluation processes enables proactive identification of risks before deployment. Such integration is particularly important as organizations increasingly rely on AI systems to produce production-level software components.

From a theoretical perspective, the study contributes to the emerging field of secure AI-assisted software engineering by connecting human-centered AI principles with cybersecurity-driven software synthesis. The framework advances existing discussions beyond vulnerability detection toward systematic prevention and governance. From a practical perspective, it offers software teams, security professionals, and organizations a structured model for deploying AI coding assistants in security-sensitive environments.

Future research should focus on empirical validation of the framework through controlled experiments, industrial case studies, and longitudinal evaluations. Additional investigation is needed to assess the effectiveness of security-aware prompting strategies, automated vulnerability assessment techniques, adaptive learning mechanisms, and explainable code generation interfaces. Research may also explore domain-specific implementations for healthcare, finance, critical infrastructure, and cloud computing environments where security requirements are particularly stringent.

In conclusion, secure AI-assisted code generation requires more than accurate code synthesis. It requires transparency, accountability, security awareness, and continuous human involvement. By embedding security heuristics and human expertise into the software synthesis process, organizations can achieve a balanced approach that leverages the strengths of artificial intelligence while mitigating its security risks. The proposed human-centric framework provides a foundation for developing trustworthy, explainable, and security-aware AI programming systems capable of supporting the future of software engineering.

REFERENCES

1. E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to Backdoor Federated Learning," in Artificial Intelligence and Statistics (AISTATS), 2020.
2. J. Brooke, "SUS: A "quick and dirty" usability scale," in Usability evaluation in industry, CRC Press, 1996, pp. 189-194.
3. M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.
4. M. D. Ernst, "The future of software engineering," in Proceedings of the 39th International Conference on Software Engineering: Future of Software Engineering Track, 2017, pp. 103-118.
5. D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. L. Dyer, D. Song, and J. Steinhardt, "Measuring Coding Challenge Competence With APPS," arXiv preprint arXiv:2105.09938, 2021.
6. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and Understanding Recurrent Networks," arXiv preprint arXiv:1506.02078, 2015.
7. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in Artificial Intelligence and Statistics (AISTATS), 2017.
8. MITRE 2023, CWE Top 25 Most Dangerous Software Weaknesses," MITRE Corporation, 2023. [Online]. Available: https://cwe.mitre.org/top25/archive/2023/2023cwetop25.html
9. R. C. Monarch, Human-in-the-Loop Machine Learning: Active Learning and Annotation for Human-Centered AI. Manning Publications, 2021.
10. N. D. Nguyen et al., "Federated Learning for Intrusion Detection: A Survey," ACM Computing Surveys, vol. 54, no. 10s, pp. 1-36, 2022.
11. OWASP, OWASP Top Ten," Open Web Application Security Project, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/
12. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot"s Code Contributions," in IEEE Symposium on Security and Privacy (S&P), 2022.
13. S. Ross, M. C. Hughes, and F. Doshi-Velez, "Right for the right reasons: Training differentiable models by constraining their explanations," in Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 2017, pp. 2662-2670.
14. M. I. Siddiq and T. F. Bissyande, "Security Smells in AI-Generated Code: An Empirical Study of GitHub Copilot," in 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2023.
15. R. Sobania, D. Briesch, M. and C.J. Karl, "An Analysis of the Security and Code Quality of AI-Coders," in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023.
16. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.