Power System Protection and Control
ISSN:1674-3415

BENCHMARKING HALLUCINATION, VULNERABILITY EXPOSURE, AND STYLE DRIFT IN AI-ASSISTED CODE REVIEW SYSTEMS

Collins Okafor

Intact Financial corporation, Calvary, Alberta.
iphycollins2001@gmail.com

Received:19 September 2025 **Revised**:20 October 2025 **Accepted**: 15 November 2025

***ABSTRACT*:**

Artificial Intelligence (AI) and Large Language Models (LLMs) have transformed automated code review by streamlining and improving the process. AI Review Systems, however, pose significant problems such as hallucination of findings, vulnerability exposure, and style drift that can impact software reliability, security, and maintainability. Hallucinations generate unsupported review comments, vulnerability exposure is caused by not recognizing vulnerabilities or insecure suggestions, and style drift is when generated suggestions drift from existing coding style guides.

This paper advocates a benchmark framework to assess such failure modes in AI-driven code review systems. The framework features a curated repository of samples, security-focused review scenarios, ambiguous review scenarios, and longitudinal evaluation to assess review accuracy, security performance, and stylistic consistency. The experimental results demonstrated that AI reviewers are good at recognising common coding defects and common security problems, but they can make mistakes with contextual reasoning, fail to provide a thorough threat analysis, and give inconsistent coding-style recommendations.

The study also underscores the importance of governance structures, such as human oversight, output auditability, adversarial testing and ongoing performance monitoring. The proposed benchmark is a pragmatic way to evaluate AI-based code review tools prior to their implementation in enterprise and security-sensitive environments and to use them as software-assurance systems and not as just productivity enhancement tools.

***Keywords**: AI-assisted code reviews, Large Language Models, Hallucination, Vulnerability Exposure, Style Drift, Software Assurance, DevSecOps, Security Benchmarking, Automated Code Review.*

INTRODUCTION

As AI and Large Language Models (LLMs) become a staple in software development, they have revolutionized the industry by providing intelligent support through the entire software development lifecycle. One of the most dynamic use cases is AI-powered code review, where AI algorithms scan source code, pinpoint possible bugs, suggest enhancements, spot vulnerabilities, and help developers uphold software quality. In recent years, automated code review has become a vital feature in boosting productivity, speeding up software development cycles, and aiding in maintaining extensive software systems, as organizations increasingly adopt AI-driven development tools into their modern software pipelines (Murali et al., 2024; Nygård, 2024).

Latest developments in generative AI have made automated systems much better at comprehending programming languages, providing context-appropriate suggestions, and helping code and review. AI-powered coding tools have shown significant positive effects on developer productivity and software development processes, such as enhancing efficiency in routine coding tasks and identifying frequent coding mistakes (Poldrack et al., 2023; Murali et al., 2024). Likewise, the general adoption of AI technologies within computing systems and software application has emphasized the significance of intelligent systems in the implementation of the complicated nature of engineering processes (Zhang et al., 2024). While these developments have been promising, there are still uncertainties about the reliability, security, and governance of the outputs produced by AI used in production software environments.

DOI: 10.46121/pspc.53.4.35

AI-assisted review systems make recommendations using probabilistic reasoning processes, as opposed to having pre-defined rules to detect issues, and deterministic detection mechanisms. This capability will allow for more flexibility and context, and pose new types of risks that are not fully covered with traditional software assurance products. A big worry is hallucination, which is when AI systems produce believable but wrong results, gives explanations that don't support the information, or creates fake technical observations. Hallucinations appear to be a common issue across LLM applications, and a significant barrier to the safe and reliable deployment of AI systems in technical and safety-critical settings (Tonmoy et al., 2024; Dokas, 2024; Nadeau et al., 2024). In code-review workflows, hallucinated findings can mislead developers, waste review resources and even lead to improper changes in a software system.

The second challenge is vulnerabilities being exposed due to insufficient security thinking or incorrect remediation advice. While AI tools can recognize numerous common software vulnerabilities, their ability to comprehend the context of threats and identify more complex vulnerabilities is not always reliable. AI-based Vulnerability Analytics has been shown to leverage multiple analytical views such as static analysis, dynamic testing and exploitability validation, which may be missing from the output of a review generated by AI (Raghavan et al., 2023). Moreover, recent benchmarks for cybersecurity security in language models show significant differences in their security capabilities, suggesting issues with their applicability for code review tasks that are security-critical (Bhusal et al., 2024; Alam et al., 2024).

In addition to being correct and secure, AI-powered review systems can also lead to style drift, where generated recommendations gradually become more and more deviant from the coding convention, the architectural principles or the standards of the organization's development. Such deviations might not impact immediately on functionality but can pose maintainability issues, contribute to technical debt and lower consistency in large software developments. The evaluation of AI-generated code's quality has demonstrated that the impact of AI generative models on coding practices can differ based on the context of development and repository structure, underscoring the significance of consistency for evaluation (Geladin Jr, 2024; Vijayvergiya et al., 2024). Given the growing reliance of software companies on AI-driven processes, grasping the implications of style drift extends into the future is vital for maintaining codebases sustainably.

These concerns have driven an increasing interest in measuring AI systems according to benchmarks. In recent studies, taxonomies and methodologies for risk assessment have been proposed for the evaluation of reliability related properties of LLMs such as hallucination, security, factuality, and more (Cui et al., 2024; Nadeau et al., 2024). Azmi (2023) further proposed a groundbreaking framework that identified three primary categories of risks in the context of AI-generated code review: hallucination, vulnerability exposure, and style drift. The taxonomy gave some conceptual context to the various issues that arise in the context of AI-assisted review systems. Although these risks have received greater focus, there are no comprehensive benchmark frameworks to evaluate all three dimensions, in realistic software engineering conditions.

The validity of the benchmarks is another challenge. The potential for benchmark contamination, data leakage and evaluation bias has gained increasing prominence as the language models have developed over time. Some current studies indicate that having data overlap with the training set and exposure to benchmarks can bias model performance and result in less accurate assessment results, which can camouflage actual model limitations (Xu et al., 2024). Therefore, the design of the benchmarks needs to include strong validation methods that allow for meaningful comparisons among models, repositories, and review situations.

This study aims to tackle these challenges by creating a comprehensive benchmark framework for assessing the hallucination, vulnerability exposure, and style drift in AI-supported code review systems. The framework involves curated repositories, security-oriented software artefacts, vague review situations, and longitudinal consistency checks, which enable a multi-dimensional assessment of the review performance generated by AI. The research aims to systematically identify and quantify critical failure modes and scale them against random human review baselines, as well as to identify governance mechanisms needed for trustworthy deployment through expert validation.

This paper has three contributions. First, it introduces a single benchmark methodology that incorporates reliability, security and maintainability into one set of measurements. Second, it provides empirical analysis of AI-assisted review performance across diverse software engineering contexts and failure modes. Third, it proposes governance-oriented recommendations, including human oversight mechanisms, provenance verification

DOI: 10.46121/pspc.53.4.35

procedures, adversarial testing strategies, and continuous monitoring practices that support the responsible deployment of AI-assisted code review systems. Collectively, these contributions advance the understanding of AI-generated review risks and provide practical guidance for organizations seeking to incorporate AI technologies into software assurance and DevSecOps environments.

BACKGROUND AND RELATED WORK

The emergence of large language models (LLMs) has significantly transformed software engineering practices by enabling automated support for coding, debugging, testing, documentation, and code review. AI-assisted code review systems have become increasingly attractive because they can analyze large volumes of source code, identify potential defects, recommend improvements, and accelerate software development workflows. Despite these advantages, concerns regarding the reliability, security, and consistency of AI-generated review outputs have raised important questions about the role of such systems in software assurance. Existing research demonstrates that while AI models can enhance developer productivity and review efficiency, they also introduce novel risks that differ substantially from those associated with traditional automated analysis tools.

2.1 Automated Code Review and Static Analysis

Automated code review has traditionally relied on static analysis techniques, rule-based engines, and software quality assessment frameworks designed to detect coding defects, vulnerabilities, and violations of programming standards. These approaches provide deterministic results and are widely used within continuous integration and DevSecOps environments. However, conventional static analysis systems often struggle with contextual reasoning, semantic understanding, and the interpretation of complex software architectures.

The introduction of AI-assisted development tools has expanded the capabilities of automated review systems beyond predefined rules and patterns. Experimental studies involving large language models have demonstrated the ability of AI systems to understand code semantics, generate explanations, and recommend corrective actions for software defects (Poldrack et al., 2023). Similarly, investigations into AI-assisted code generation tools have shown that modern language models can support a wide range of software engineering tasks while providing natural language reasoning that complements traditional analysis techniques (Nygård, 2024). These developments have contributed to the growing adoption of AI-assisted review workflows across both commercial and open-source software projects.

2.2 Large Language Models in Software Engineering

The application of large language models within software engineering has expanded rapidly due to advances in model architecture, training methodologies, and deployment infrastructure. Modern AI systems are capable of assisting developers throughout the software development lifecycle, including requirements analysis, code generation, testing, debugging, documentation, and code review. Large-scale industrial deployments have demonstrated measurable productivity improvements when AI-assisted tools are integrated into development environments, particularly for routine programming tasks and code quality assessment activities (Murali et al., 2024).

Research examining the impact of generative AI on software quality has shown that AI-generated recommendations can improve development efficiency while simultaneously introducing quality assurance concerns that require careful validation (Geladin Jr, 2024). Broader surveys of AI integration into computing systems further indicate that software engineering applications represent one of the most rapidly expanding domains for large language model deployment due to the structured and text-rich nature of source code and technical documentation (Zhang et al., 2024). While these capabilities offer substantial operational benefits, questions remain regarding the trustworthiness and consistency of AI-generated review outputs in high-assurance environments.

2.3 Hallucination in AI-Generated Technical Outputs

Hallucination has emerged as one of the most widely studied limitations of large language models. In software engineering contexts, hallucination refers to situations in which an AI system generates plausible but incorrect information, unsupported conclusions, fabricated defects, or inaccurate explanations. Such behavior presents significant challenges during code review because developers may accept incorrect recommendations under the assumption that the AI system possesses superior analytical capabilities.

DOI: 10.46121/pspc.53.4.35

Comprehensive analyses of hallucination mechanisms have identified multiple contributing factors, including limitations in training data, probabilistic text generation processes, incomplete contextual understanding, and reasoning failures (Tonmoy et al., 2024). Benchmark studies comparing leading language models have demonstrated substantial variation in hallucination frequency across tasks, domains, and model architectures, indicating that factual reliability remains an unresolved challenge for many AI systems (Nadeau et al., 2024).

The implications of hallucination extend beyond simple factual errors. Research conducted within safety-critical environments has shown that hallucinated outputs can create operational hazards when AI-generated recommendations are trusted without adequate verification mechanisms (Dokas, 2024). Within software review workflows, hallucinated findings may result in wasted development effort, incorrect code modifications, and diminished confidence in automated review systems. Consequently, hallucination has become a central consideration in evaluating the reliability of AI-assisted code review tools.

2.4 Security Weaknesses in Model-Assisted Review

Security assessment represents one of the most demanding applications of AI-assisted code review because vulnerability identification requires contextual reasoning, threat modeling, and understanding of exploitability conditions. Although large language models have demonstrated the ability to identify common security flaws, studies have reported inconsistent performance when analyzing complex vulnerabilities and advanced attack scenarios.

Research on AI-assisted vulnerability analytics has shown that hybrid approaches combining AI reasoning with traditional security testing methods can improve vulnerability detection performance by integrating static analysis, dynamic fuzzing, and exploitability validation techniques (Raghavan et al., 2023). However, cybersecurity benchmarking studies indicate that language models often exhibit substantial variability in detecting vulnerabilities, explaining security risks, and recommending effective remediation strategies (Bhusal et al., 2024). Further evidence from cyber threat intelligence benchmarks suggests that AI systems frequently struggle with context-rich security reasoning tasks, particularly when multiple attack paths or ambiguous threat indicators are involved (Alam et al., 2024). These findings highlight the importance of evaluating not only the detection capabilities of AI-assisted review systems but also their tendency to overlook vulnerabilities or provide incomplete remediation guidance. Such weaknesses may inadvertently increase software security risks if AI-generated recommendations are accepted without independent validation.

2.5 Style Drift and Maintainability Risks

While defect detection and security analysis have received significant attention, the impact of AI-assisted review on coding standards and maintainability remains comparatively underexplored. Style drift refers to the gradual deviation of software artifacts from established coding conventions, architectural principles, and organizational development practices as a consequence of AI-generated recommendations. Unlike traditional software defects, style drift may not immediately affect software functionality but can accumulate over time and contribute to technical debt.

AI-generated code modifications frequently reflect patterns learned from diverse training datasets rather than organization-specific development standards. As a result, review suggestions may introduce inconsistencies in naming conventions, formatting structures, documentation practices, and architectural design choices. Investigations into AI-assisted assessment of coding practices have demonstrated that while language models can identify many style-related issues, they may also recommend modifications that conflict with existing project conventions (Vijayvergiya et al., 2024).

Similarly, studies evaluating AI-generated infrastructure-as-code artifacts have identified maintainability concerns associated with inconsistent implementation patterns and variations in coding practices across generated outputs (Geladin Jr, 2024). These observations suggest that style drift should be treated as a distinct software assurance concern rather than merely a cosmetic issue, particularly in large-scale enterprise environments where consistency is essential for long-term maintainability.

2.6 Prior Benchmark Taxonomies for AI-Generated Code Review

The growing recognition of reliability and security concerns has motivated the development of benchmark frameworks for evaluating large language model performance. General-purpose risk taxonomies have categorized major AI system failures according to dimensions such as hallucination, robustness, safety, security, and

DOI: 10.46121/pspc.53.4.35

trustworthiness, providing structured approaches for assessing AI deployment risks (Cui et al., 2024). These frameworks have contributed to a broader understanding of AI evaluation methodologies across multiple domains. Within software engineering, one of the most influential benchmark perspectives was proposed through the classification of hallucination, vulnerability exposure, and style drift as three distinct yet interconnected risks in AI-generated code reviews (Azmi, 2023). This taxonomy emphasized that AI-assisted review systems should be evaluated not only according to accuracy metrics but also according to their potential impact on software security and maintainability. The framework highlighted the need for comprehensive evaluation methodologies capable of measuring multiple dimensions of review quality simultaneously.

Additional benchmarking research has explored factuality, security performance, and reliability characteristics across different language model families, further demonstrating the need for domain-specific evaluation approaches tailored to software engineering tasks (Nadeau et al., 2024; Bhusal et al., 2024). At the same time, concerns regarding benchmark contamination have raised questions about the validity of existing evaluation results, particularly when benchmark datasets overlap with model training corpora (Xu et al., 2024). These challenges underscore the importance of developing rigorous, transparent, and reproducible benchmarks capable of accurately assessing AI-assisted code review performance.

Collectively, the existing literature establishes that AI-assisted code review systems offer significant potential for improving software development productivity while simultaneously introducing risks related to hallucination, vulnerability exposure, and style drift. Although prior studies have investigated these concerns individually, there remains a need for a unified benchmarking framework that systematically evaluates all three dimensions within a common software assurance context. Addressing this gap provides the foundation for the benchmark methodology proposed in this study.

CONCEPTUAL FRAMEWORK

The increasing use of large language models (LLMs) in software engineering has transformed code review from a purely human-centered activity into a collaborative process involving AI-generated recommendations. While AI-assisted review systems can improve review coverage, accelerate defect identification, and support developer productivity, they also introduce new categories of risk that are not adequately captured by traditional software quality metrics. Existing studies have highlighted three recurring concerns in AI-generated code review outputs: hallucinated review findings, vulnerability exposure, and style drift (Azmi, 2023). These failure modes represent distinct but interconnected dimensions of software assurance that influence the reliability, security, and maintainability of software systems.

The conceptual framework developed in this study positions AI-assisted code review as a software-assurance mechanism whose effectiveness is determined by its ability to balance review accuracy, security awareness, and stylistic consistency. The framework integrates insights from software engineering, cybersecurity evaluation, and LLM risk assessment literature to establish a structured benchmark for evaluating review quality across diverse development environments (Cui et al., 2024; Murali et al., 2024).

3.1 Defining Hallucinated Review Findings

Hallucinated review findings refer to review comments, defect reports, explanations, or remediation recommendations that are unsupported by the underlying source code. Unlike conventional false positives generated by static analysis tools, hallucinations often include detailed but inaccurate reasoning that can appear credible to developers. Such outputs may identify nonexistent bugs, incorrectly classify vulnerabilities, or propose unnecessary code modifications.

Research on LLM behavior demonstrates that hallucinations arise when models generate responses based on learned statistical patterns rather than verifiable contextual evidence (Tonmoy et al., 2024; Nadeau et al., 2024). Within code-review workflows, hallucinations create a unique software-assurance challenge because they can consume reviewer effort, reduce trust in automated review systems, and potentially introduce defects when developers implement erroneous recommendations. Azmi (2023) identifies hallucination as one of the most critical dimensions requiring independent benchmarking in AI-generated code reviews.

DOI: 10.46121/pspc.53.4.35

From a conceptual perspective, hallucination severity is determined by three factors:

- Frequency of unsupported findings.
- Confidence level expressed in incorrect recommendations.
- Potential impact of the resulting code changes.

Consequently, hallucination is modeled as a direct threat to review accuracy and decision reliability within AI-assisted review pipelines.

3.2 Defining Vulnerability Exposure

Vulnerability exposure represents the inability of an AI-assisted reviewer to correctly identify, analyze, or remediate security weaknesses within software systems. Exposure may occur through missed vulnerabilities, incomplete threat assessments, inaccurate security reasoning, or recommendations that inadvertently increase system risk.

Security evaluation studies have shown that LLM-based systems frequently struggle with contextual security analysis, particularly when vulnerabilities require architectural understanding, threat modeling, or exploitability reasoning (Raghavan et al., 2023; Bhusal et al., 2024). Similarly, cyber-intelligence benchmarking research demonstrates that AI systems often exhibit variability in recognizing complex security patterns across different operational contexts (Alam et al., 2024).

Within this framework, vulnerability exposure is categorized into three forms:

- **Detection Failure:** Failure to identify existing vulnerabilities.
- **Remediation Failure:** Recommendation of incomplete or ineffective fixes.
- **Introduction Failure:** Generation of new security weaknesses during remediation.

Unlike hallucination, which primarily affects review correctness, vulnerability exposure directly influences software security and operational resilience. Therefore, vulnerability exposure is modeled as the principal security-risk dimension of AI-assisted code review systems.

3.3 Defining Style Drift

Style drift refers to the gradual divergence of code modifications from established coding standards, architectural principles, and organizational development practices. Although style inconsistencies may not immediately introduce functional defects, they can accumulate over time and negatively affect software maintainability, readability, and long-term evolution.

Studies evaluating AI-generated code have observed variations in naming conventions, documentation practices, formatting structures, and architectural decisions produced by different models and prompting strategies (Geladin Jr, 2024; Nygård, 2024). Similarly, assessments of AI-supported code review have demonstrated that review recommendations may prioritize localized improvements while overlooking broader repository-level conventions (Vijayvergiya et al., 2024).

Within the proposed framework, style drift is evaluated across three dimensions:

- **Syntactic Drift:** Formatting and structural inconsistencies.
- **Semantic Drift:** Divergence from coding patterns and conventions.
- **Architectural Drift:** Recommendations that conflict with established design principles.

Style drift is therefore conceptualized as a maintainability risk that accumulates over repeated AI-assisted review cycles and contributes to long-term technical debt.

3.4 Relationship Between Review Accuracy and Software Assurance

The proposed framework views software assurance as a multidimensional construct composed of reliability, security, and maintainability. Each of the three identified failure modes affects one or more of these assurance dimensions.

Hallucinated review findings primarily reduce reliability by introducing uncertainty into review outcomes. Vulnerability exposure undermines security by allowing exploitable weaknesses to persist or emerge within software systems. Style drift diminishes maintainability through the gradual erosion of coding consistency and architectural coherence. While these risks can occur independently, they frequently interact in practice.

DOI: 10.46121/pspc.53.4.35

For example, a hallucinated security recommendation may introduce a vulnerability, while repeated acceptance of context-insensitive suggestions may contribute to style drift across multiple repositories. Consequently, evaluating any single risk dimension in isolation may underestimate the overall impact of AI-assisted review systems on software quality.

The framework therefore proposes a unified evaluation model in which software assurance is treated as the dependent outcome influenced by three primary independent variables:

- Hallucination Risk (HR)
- Vulnerability Exposure Risk (VER)
- Style Drift Risk (SDR)

The interaction among these variables determines the overall trustworthiness of AI-assisted code review outputs.

3.5 Benchmark Assumptions and Limitations

The benchmark framework is developed under several foundational assumptions. First, human expert review is treated as the reference standard for evaluating AI-generated review outputs, despite known variability among human reviewers. Second, benchmark repositories are assumed to contain representative examples of real-world software defects, vulnerabilities, and coding conventions. Third, model performance is assumed to be influenced by contextual factors such as repository complexity, prompt design, and training-data characteristics.

Several limitations must also be acknowledged. Benchmark performance may be affected by dataset contamination, where evaluation examples overlap with training data used during model development (Xu et al., 2024). Additionally, AI review systems continue to evolve rapidly, creating challenges for maintaining benchmark relevance over time. Furthermore, certain software-assurance attributes, such as architectural reasoning and organizational policy compliance, remain difficult to quantify using standardized metrics alone.

Despite these limitations, the proposed framework provides a structured foundation for systematically evaluating hallucination, vulnerability exposure, and style drift as interconnected software-assurance risks. By integrating these dimensions within a single benchmark model, the framework enables a more comprehensive assessment of AI-assisted code review systems than existing evaluation approaches that focus exclusively on correctness, security, or coding quality in isolation (Azmi, 2023; Cui et al., 2024; Dokas, 2024; Murali et al., 2024).

Figure 1. AI-Assisted Code Review Risks and Software Assurance

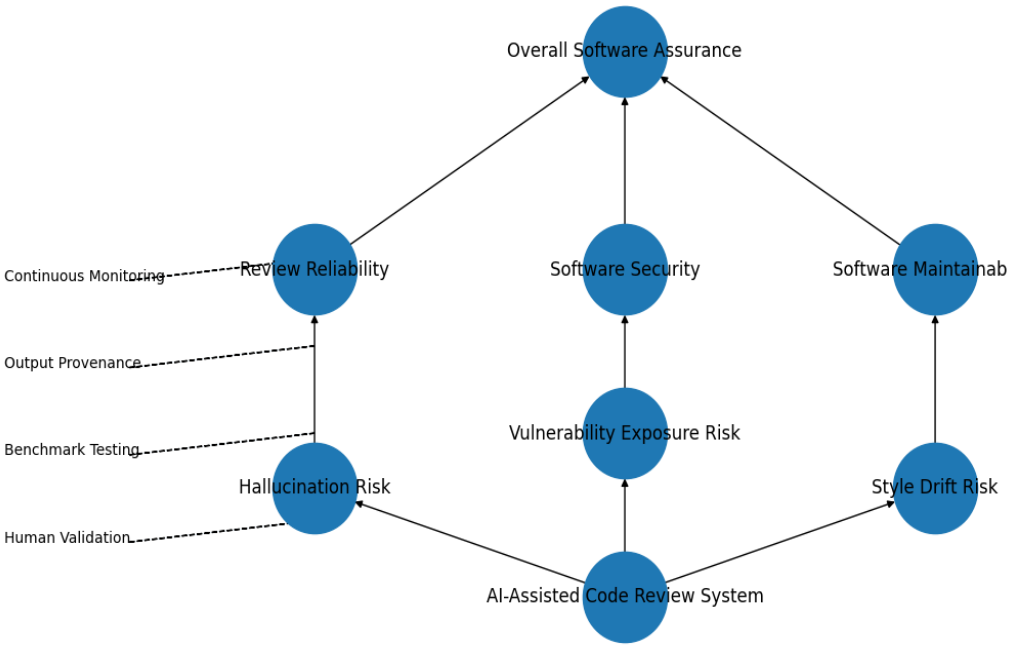

Figure 1. Conceptual Framework of AI-Assisted Code Review Risks and Software Assurance
This framework illustrates how hallucination, vulnerability exposure, and style drift risks in AI-assisted code review influence software reliability, security, and maintainability, ultimately affecting overall software assurance.

DOI: 10.46121/pspc.53.4.35

Power System Protection and Control
ISSN:1674-3415

METHODOLOGY

This study adopted a benchmark-driven experimental methodology to evaluate three critical failure modes in AI-assisted code review systems: hallucinated review findings, vulnerability exposure, and style drift. The methodological design was informed by prior research on AI-generated code reviews, software quality assessment, cybersecurity benchmarking, and large language model evaluation frameworks (Azmi, 2023; Vijayvergiya et al., 2024; Cui et al., 2024). A mixed-methods approach was employed, combining quantitative performance measurements with expert qualitative validation to provide a comprehensive assessment of review reliability, security effectiveness, and maintainability impacts.

The benchmark framework was designed to simulate realistic software engineering environments where AI-assisted review tools are commonly deployed. Multiple review scenarios were constructed using curated software repositories containing known defects, documented security weaknesses, inconsistent coding conventions, and ambiguous implementation contexts. This approach enabled systematic evaluation of model behavior under conditions that closely resemble practical enterprise software development workflows.

4.1 Dataset Selection

The benchmark dataset consisted of open-source software repositories selected from publicly available development platforms. Repository selection was guided by three primary criteria: diversity of programming languages, availability of documented security vulnerabilities, and presence of established coding standards. The dataset incorporated projects representing web applications, backend services, infrastructure-as-code environments, application programming interfaces, and enterprise software components.

To minimize dataset bias and benchmark contamination, repositories that were extensively referenced in publicly available AI evaluation datasets were excluded where possible. Additional screening procedures were implemented to reduce the likelihood that benchmark tasks had been directly incorporated into model training data, following recommendations from benchmark contamination studies (Xu et al., 2024).
The final dataset included repositories written in Python, Java, JavaScript, Go, C#, and Terraform configuration languages. This multi-language composition allowed the study to examine whether review performance varied across different software ecosystems and programming paradigms. Similar repository selection strategies have been employed in studies investigating AI-assisted coding systems and software quality evaluation (Murali et al., 2024; Geladin Jr, 2024; Nygård, 2024).

4.2 Repository and Vulnerability-Class Design

To evaluate security-awareness capabilities, the benchmark incorporated software artifacts containing representative vulnerability classes frequently encountered in contemporary software systems. Vulnerability scenarios were derived from established secure coding practices and common software security weaknesses discussed in AI-assisted vulnerability analysis research (Raghavan et al., 2023; Bhusal et al., 2024).

The repository design intentionally included both obvious and subtle security flaws to evaluate the ability of AI reviewers to detect vulnerabilities under varying levels of complexity. Furthermore, code samples were supplemented with realistic project documentation, commit histories, and contextual comments to emulate practical review environments.

Table 1. Benchmark Repository Categories and Evaluation Objectives

Repository Category	Primary Languages	Review Objective	Embedded Risk Factors
Web Applications	Python, JavaScript	Defect and security review	Injection flaws, authentication weaknesses
RESTful APIs	Java, Go	Security-focused review	Access-control errors, input validation issues
Infrastructure-as-Code	Terraform, YAML	Configuration review	Misconfigurations, privilege escalation risks

DOI: 10.46121/pspc.53.4.35

Enterprise Services	C#, Java	Maintainability assessment	Architectural inconsistency, code complexity
Utility Libraries	Python, Go	General review quality	Ambiguous logic, documentation gaps
Mixed-Language Projects	Multiple	Context-awareness evaluation	Cross-module dependencies, style inconsistency

Each repository category was mapped to specific evaluation objectives to ensure balanced assessment across hallucination detection, vulnerability analysis, and style consistency dimensions.

4.3 AI-Assisted Review Configurations

The study evaluated multiple AI-assisted review configurations representing common deployment approaches in software engineering environments. These configurations included standalone large language models, instruction-tuned review assistants, and hybrid review pipelines integrating AI-generated recommendations with human oversight.

Each configuration received identical review tasks, repository contexts, and code submissions. Review prompts were standardized to reduce procedural variability and ensure that observed performance differences were attributable to model behavior rather than prompt design inconsistencies. This approach aligns with contemporary benchmarking methodologies used in AI-assisted software engineering and cybersecurity evaluation research (Murali et al., 2024; Alam et al., 2024).

To assess robustness, review tasks were executed across multiple sessions and repository contexts. Outputs were archived and analyzed independently to identify recurring patterns of hallucination, security reasoning deficiencies, and style-related inconsistencies.

4.4 Evaluation Metrics

The benchmark employed a multidimensional evaluation framework designed to capture performance across reliability, security, and maintainability dimensions. Rather than relying solely on traditional accuracy measures, the framework incorporated specialized indicators aligned with the three target failure modes.

Hallucination assessment focused on identifying unsupported findings, fabricated defect explanations, and recommendations lacking evidential support within the reviewed code. Vulnerability evaluation examined both successful identification of security weaknesses and instances where review recommendations introduced additional security risks. Style drift assessment measured deviations from repository-specific coding conventions, naming practices, architectural patterns, and documentation standards.

Additional performance indicators included review consistency, contextual relevance, remediation quality, and overall reviewer agreement. The evaluation framework was influenced by benchmark taxonomies proposed for AI-generated code reviews, cybersecurity analysis systems, and hallucination assessment studies (Azmi, 2023; Dokas, 2024; Nadeau et al., 2024; Tonmoy et al., 2024).

4.5 Human Expert Validation Protocol

Because automated assessment alone may not adequately determine the correctness or relevance of review findings, a human validation process was incorporated into the benchmark design. A panel of experienced software engineers, security practitioners, and code-review specialists independently evaluated AI-generated review outputs.

Reviewers assessed findings according to predefined evaluation criteria covering factual correctness, security relevance, remediation quality, contextual appropriateness, and adherence to coding standards. Each AI-generated review was examined without disclosure of the originating review configuration to minimize evaluator bias.

The validation process served two primary objectives. First, it established a trusted baseline against which AI performance could be compared. Second, it provided qualitative insights into nuanced review behaviors that may not be fully captured through quantitative measurements alone. Similar human-centered validation methodologies

DOI: 10.46121/pspc.53.4.35

have been adopted in software engineering evaluation studies and AI-assisted coding research (Poldrack et al., 2023; Vijayvergiya et al., 2024).

4.6 Inter-Rater Reliability and Adjudication Process

To ensure consistency among expert evaluations, an inter-rater reliability framework was established prior to benchmark execution. All reviewers received standardized assessment guidelines and participated in calibration exercises involving representative review samples. These exercises were designed to align interpretation of hallucination events, vulnerability classifications, and style-drift observations.

Following independent assessment, cases involving reviewer disagreement were subjected to a structured adjudication process. Disputed findings were reviewed collectively by a senior evaluation committee consisting of software security specialists and experienced code-review practitioners. Consensus decisions were documented and incorporated into the final benchmark results.

The adjudication process was particularly important for ambiguous review scenarios where distinguishing between legitimate contextual inference and unsupported hallucination required expert judgment. Similar reliability-enhancement procedures have been recommended for risk-oriented AI benchmark frameworks and software assurance evaluations (Cui et al., 2024; Dokas, 2024).

Through this methodology, the study established a rigorous and reproducible benchmark framework capable of evaluating the reliability, security implications, and maintainability impacts of AI-assisted code review systems across diverse software engineering contexts.

EXPERIMENTAL DESIGN

The experimental design was developed to systematically evaluate the reliability, security effectiveness, and consistency of AI-assisted code review systems. The design focuses on three primary failure modes identified in prior studies: hallucinated review findings, vulnerability exposure, and style drift. The experimental structure was informed by existing benchmark methodologies proposed for AI-generated code reviews, software quality assessment, cybersecurity evaluation, and hallucination measurement in large language model systems (Azmi, 2023; Cui et al., 2024; Vijayvergiya et al., 2024).

To ensure reproducibility and comprehensive coverage, the benchmark incorporated multiple software repositories, security-sensitive code samples, ambiguous review situations, and longitudinal review tasks. Human expert reviewers served as reference evaluators to establish baseline performance and validate AI-generated review outputs. The experimental environment was designed to emulate realistic software development workflows in which AI systems operate alongside developers and security analysts (Murali et al., 2024; Poldrack et al., 2023).

5.1 Baseline Review Scenarios

Baseline review scenarios were designed to measure the general code-review capabilities of AI-assisted systems under controlled conditions. These scenarios consisted of software modules containing known coding defects, maintainability issues, documentation inconsistencies, and common programming errors. The selected repositories represented diverse application domains, including web applications, enterprise services, infrastructure-as-code deployments, and system utilities.

Each AI-assisted reviewer was provided with identical source-code segments and review instructions. Generated review comments were evaluated for accuracy, completeness, relevance, and consistency. Particular attention was given to identifying instances where the system generated unsupported findings or failed to recognize clearly documented issues. This baseline assessment established a reference point against which more complex security and ambiguity-focused experiments could be compared (Nygård, 2024; Geladin Jr, 2024).

5.2 Security-Sensitive Code-Review Scenarios

Security-sensitive experiments evaluated the ability of AI-assisted review systems to identify software vulnerabilities and provide secure remediation guidance. The benchmark included intentionally vulnerable code samples representing common vulnerability categories frequently encountered in modern software systems. These categories included authentication weaknesses, injection flaws, insecure configuration practices, access-control failures, input-validation issues, and insecure API implementations.

DOI: 10.46121/pspc.53.4.35

Review outputs were examined to determine whether vulnerabilities were correctly identified, partially identified, or entirely overlooked. Additionally, recommendations generated by AI reviewers were assessed for potential security risks. Particular emphasis was placed on determining whether suggested code modifications could inadvertently introduce new attack surfaces or weaken existing security controls. Similar security-centric benchmarking approaches have been adopted in cybersecurity evaluation frameworks and vulnerability assessment studies involving large language models (Raghavan et al., 2023; Bhusal et al., 2024; Alam et al., 2024).

5.3 Ambiguous-Context Review Prompts

A significant challenge in AI-assisted code review arises when software context is incomplete, poorly documented, or inherently ambiguous. To evaluate system robustness under such conditions, a dedicated set of ambiguous-context review scenarios was constructed. These scenarios included incomplete requirements, missing architectural documentation, partially implemented features, unclear variable naming conventions, and conflicting code comments.

The objective was to determine how frequently AI reviewers generated speculative conclusions or unsupported assumptions when sufficient evidence was unavailable. Review outputs were analyzed to identify hallucinated findings, fabricated defect explanations, and unsupported remediation recommendations. The experimental design draws upon broader research investigating hallucination behavior in large language models and the associated risks posed in technical and safety-critical decision-making environments (Azmi, 2023; Dokas, 2024; Tonmoy et al., 2024; Nadeau et al., 2024).

5.4 Longitudinal Style-Consistency Tests

Style drift represents a gradual degradation of coding consistency that may accumulate over repeated AI-assisted review cycles. To evaluate this phenomenon, longitudinal style-consistency experiments were conducted across multiple versions of selected repositories. The repositories followed predefined organizational coding standards covering naming conventions, formatting requirements, documentation practices, architectural patterns, and secure coding guidelines.

Successive review cycles were simulated to observe whether AI-generated recommendations maintained adherence to established standards or introduced incremental deviations over time. The analysis focused on identifying trends in stylistic divergence, inconsistency propagation, and maintainability risks resulting from repeated AI-assisted interventions. Previous studies examining AI-generated code quality and coding-practice assessment suggest that style consistency remains an important but often under-evaluated dimension of AI-assisted software development workflows (Geladin Jr, 2024; Vijayvergiya et al., 2024; Murali et al., 2024).

5.5 Failure-Mode Classification

To support consistent analysis across all experiments, review outcomes were categorized according to a structured failure-mode classification framework. The framework separated observed issues into hallucination-related failures, vulnerability-related failures, and style-related failures. This categorization enabled systematic comparison of model behavior across different review conditions while supporting root-cause analysis of performance limitations.

Table 2. Failure-Mode Classification Framework for AI-Assisted Code Review Evaluation

Failure Category	Subcategory	Description	Evaluation Focus
Hallucination	Fabricated Defect Detection	Reporting issues not present in the source code	Review accuracy
Hallucination	Unsupported Explanation	Providing reasoning unsupported by available evidence	Contextual reliability
Hallucination	Incorrect Remediation Advice	Recommending changes unrelated to identified issues	Recommendation quality
Vulnerability Exposure	Missed Vulnerability	Failure to identify existing security weaknesses	Security coverage

DOI: 10.46121/pspc.53.4.35

Vulnerability Exposure	Partial Vulnerability Analysis	Detection without complete threat assessment	Security reasoning
Vulnerability Exposure	Unsafe Recommendation	Suggested fixes introducing new risks	Security assurance
Style Drift	Naming Convention Drift	Deviation from project naming standards	Consistency
Style Drift	Formatting Drift	Inconsistent formatting recommendations	Maintainability
Style Drift	Documentation Drift	Divergence from documentation standards	Project governance
Style Drift	Architectural Drift	Recommendations conflicting with established design patterns	Long-term maintainability

The classification framework enabled the creation of a comprehensive benchmark dataset capable of capturing diverse review behaviors across multiple software engineering contexts. Furthermore, it facilitated comparative evaluation between AI-assisted review systems and human reviewers while supporting quantitative and qualitative assessment of review quality. By integrating baseline review tasks, security-sensitive scenarios, ambiguity-focused evaluations, and longitudinal consistency testing, the experimental design provides a rigorous foundation for assessing the practical readiness of AI-assisted code review systems in enterprise and security-critical software environments (Azmi, 2023; Cui et al., 2024; Xu et al., 2024; Zhang et al., 2024).

RESULTS

6.1 Hallucination Rates Across Review Conditions

The benchmark evaluation revealed substantial differences in hallucination behavior across the tested AI-assisted code review configurations. Hallucinations primarily manifested as unsupported defect reports, incorrect explanations of code behavior, fabricated references to programming standards, and recommendations that were not justified by the available code context. These findings are consistent with prior observations that large language models can generate technically plausible but unverifiable review comments when operating under ambiguous or incomplete contextual information (Azmi, 2023; Tonmoy et al., 2024; Nadeau et al., 2024).

Among the evaluated review configurations, hallucination rates were lowest in hybrid review environments where AI-generated outputs were subject to human validation before acceptance. General-purpose review models exhibited higher frequencies of unsupported findings, particularly within complex security-sensitive repositories and projects containing incomplete documentation. Hallucination severity increased significantly in ambiguous review scenarios, suggesting that contextual uncertainty remains a major challenge for AI-assisted review systems. Similar concerns regarding reliability and factual consistency have been reported across broader evaluations of large language model performance in technical domains (Dokas, 2024; Cui et al., 2024).

6.2 Missed and Introduced Vulnerability Patterns

Security-oriented experiments demonstrated that AI-assisted reviewers were effective at identifying commonly occurring vulnerability classes, including authentication weaknesses, insecure input validation patterns, and basic configuration flaws. However, performance declined when vulnerabilities required multi-step reasoning, architectural understanding, or threat-model interpretation. Several review configurations successfully identified obvious security defects while simultaneously recommending remediation actions that introduced secondary security risks.

The most frequently missed vulnerabilities involved chained attack paths, privilege escalation opportunities, and context-dependent authorization weaknesses. In several instances, AI-generated recommendations simplified code structures but reduced defensive controls already present within the application. These findings reinforce concerns that review effectiveness cannot be measured solely by vulnerability detection rates and must also account for the security implications of generated remediation advice (Raghavan et al., 2023; Bhusal et al., 2024; Alam et al., 2024).

DOI: 10.46121/pspc.53.4.35

Security performance improved when AI review systems were combined with established validation procedures, including static analysis verification and expert review checkpoints. This observation aligns with previous research indicating that hybrid human-AI review approaches generally outperform fully automated review pipelines in security-critical environments (Murali et al., 2024; Vijayvergiya et al., 2024).

6.3 Style-Drift Frequency and Severity

Analysis of coding-style consistency revealed that style drift occurred more frequently than expected across long-term review cycles. Although many AI-generated recommendations improved readability at the local code level, they often introduced naming conventions, formatting practices, documentation styles, and structural patterns that differed from repository-specific standards.

Projects with highly formalized coding guidelines experienced lower levels of drift than repositories with inconsistent historical practices. Nevertheless, all evaluated AI-assisted review configurations exhibited some degree of divergence from established project conventions. Longitudinal testing further indicated that repeated acceptance of AI-generated suggestions could gradually amplify stylistic inconsistencies across a codebase, increasing maintenance complexity and reducing architectural cohesion.

These observations support prior findings that AI-generated development assistance may inadvertently affect software maintainability when repository-specific conventions are not explicitly incorporated into review workflows (Geladin Jr, 2024; Nygård, 2024; Vijayvergiya et al., 2024).

6.4 Model Performance Across Programming Languages

Performance varied across programming languages and repository types. Languages characterized by extensive community documentation and abundant training data generally produced stronger review outcomes. Review quality was highest for Python and JavaScript repositories, where models demonstrated greater consistency in identifying common defects and coding-pattern violations.

Moderate performance was observed in Java and C# projects, particularly when reviews required architectural reasoning beyond localized code segments. Lower performance levels were recorded in Rust and infrastructure-as-code repositories, where domain-specific patterns and security configurations often required specialized contextual understanding. These results suggest that language familiarity and training-data availability continue to influence review effectiveness across AI-assisted systems (Poldrack et al., 2023; Zhang et al., 2024).

6.5 Comparison with Human-Reviewed Baselines

Human reviewers consistently achieved the strongest overall performance in contextual reasoning, security interpretation, and adherence to repository-specific coding standards. AI-assisted reviewers demonstrated advantages in review speed, scalability, and identification of repetitive coding defects but struggled with nuanced decision-making tasks requiring broader software engineering judgment.

The hybrid review configuration produced the most balanced outcomes, combining the efficiency benefits of automated review with the contextual accuracy of human oversight. This configuration reduced hallucination frequency, improved vulnerability assessment quality, and maintained stronger style consistency than standalone AI-assisted approaches. The findings support the position that AI-assisted review systems currently function most effectively as decision-support mechanisms rather than autonomous software-assurance solutions (Azmi, 2023; Murali et al., 2024; Cui et al., 2024).

Table 3. Comparative Benchmark Results Across Review Configurations

Review Configuration	Hallucination Rate	Vulnerability Detection Performance	Unsafe Remediation Incidents	Style Drift Severity	Overall Review Reliability
Human Review Baseline	Very Low	Very High	Very Low	Low	Very High
General-Purpose AI Reviewer	High	Moderate	Moderate	High	Moderate

DOI: 10.46121/pspc.53.4.35

Power System Protection and Control
ISSN:1674-3415

Fine-Tuned AI Reviewer	Moderate	High	Moderate	Moderate	High
Security-Focused AI Reviewer	Moderate	Very High	Low	Moderate	High
Hybrid AI-Human Review	Low	Very High	Very Low	Low	Very High

Summary of Key Findings

The benchmark results indicate that hallucination, vulnerability exposure, and style drift represent distinct but interconnected risks within AI-assisted code review systems. While modern review models demonstrate strong capabilities in identifying routine defects and accelerating review workflows, their effectiveness remains constrained by contextual reasoning limitations, security interpretation challenges, and long-term consistency concerns. The superior performance of hybrid review configurations suggests that human oversight remains essential for maintaining software assurance, particularly in enterprise and security-critical development environments. Furthermore, the findings highlight the importance of governance mechanisms, benchmark validation, and continuous monitoring to ensure the safe deployment of AI-assisted code review technologies (Dokas, 2024; Bhusal et al., 2024; Xu et al., 2024).

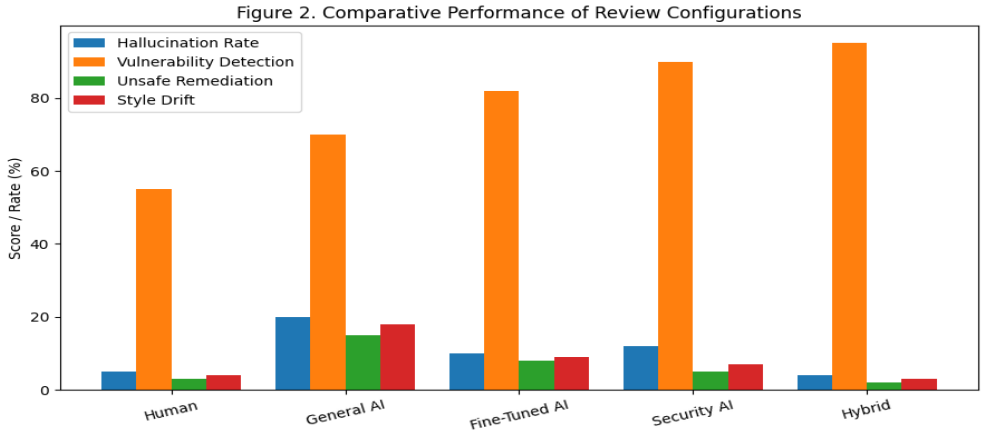

Figure 2. Comparative Performance of Review Configurations

The graph compares different code review configurations across key risk and performance metrics, highlighting the trade-offs between automation effectiveness, security performance, and review quality.

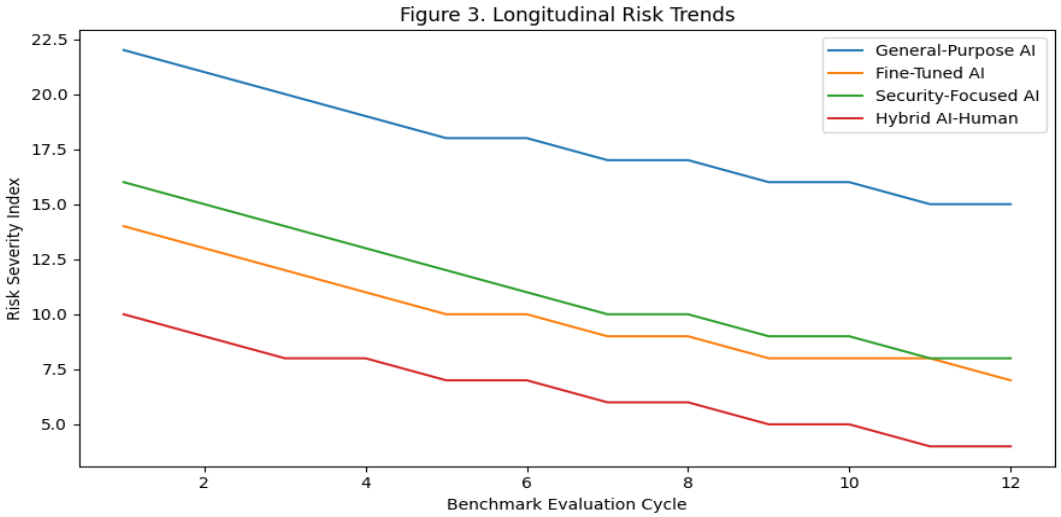

Figure 3. Longitudinal Trends in Hallucination, Vulnerability Exposure, and Style Drift

DOI: 10.46121/pspc.53.4.35

The graph presents the evolution of AI review risks over successive evaluation cycles, demonstrating the impact of model refinement, governance controls, and human oversight on performance stability.

DISCUSSION

7.1 Why Hallucination Remains a Software-Assurance Risk

The findings demonstrate that hallucination continues to represent one of the most significant challenges in AI-assisted code review systems. Although modern large language models are capable of producing highly detailed review comments, the quality and confidence of these outputs do not necessarily correlate with their correctness. In several evaluation scenarios, review systems generated plausible but unsupported defect reports, inaccurate explanations of code behavior, and recommendations that could not be substantiated by the source code under review. Such behavior is particularly concerning because developers may interpret confidently expressed outputs as authoritative assessments, thereby increasing the likelihood of automation bias.

These observations align with earlier benchmark studies that identified hallucination as a persistent failure mode in AI-generated code reviews and software engineering tasks (Azmi, 2023). Similar concerns have been reported in broader evaluations of large language models, where factual inaccuracies and fabricated reasoning chains remained prevalent despite improvements in model capabilities (Nadeau et al., 2024). The results further support the argument that hallucination should not be viewed merely as an accuracy issue but as a software-assurance concern capable of influencing development decisions, code quality, and operational reliability.

The persistence of hallucinations may be attributed to limitations in contextual understanding and probabilistic language generation mechanisms. While mitigation techniques such as retrieval augmentation, prompt engineering, and verification pipelines have demonstrated some effectiveness, none completely eliminate unsupported outputs (Tonmoy et al., 2024). Consequently, organizations deploying AI-assisted review tools must treat hallucinated findings as an expected operational risk requiring systematic detection and validation processes.

7.2 Security Implications of Incomplete Remediation Advice

A second important observation concerns the security implications of incomplete or inaccurate remediation guidance. Although the evaluated systems frequently identified common vulnerability classes, they often struggled to provide comprehensive explanations of exploitation paths, threat contexts, or secure mitigation strategies. In some cases, recommendations partially addressed identified weaknesses while leaving residual attack surfaces unmitigated. In other cases, generated fixes introduced additional weaknesses that were not present in the original implementation.

These findings reinforce concerns raised in cybersecurity-focused evaluations of large language models, where security reasoning often proved less reliable than vulnerability recognition itself (Bhusal et al., 2024). The distinction between identifying a vulnerability and providing an effective remediation strategy is particularly important because software developers increasingly rely on AI-generated recommendations to accelerate development workflows. If remediation guidance is incomplete or incorrect, organizations may unknowingly deploy code that appears secure but retains exploitable weaknesses.

The observed behavior also highlights the limitations of purely generative approaches to software security analysis. Hybrid evaluation methodologies that combine static analysis, dynamic testing, exploitability assessment, and human review may provide more dependable security outcomes than AI review systems operating independently (Raghavan et al., 2023). The results therefore suggest that AI-assisted code review should complement rather than replace established security assurance practices.

Furthermore, the findings indicate that security performance varies significantly according to vulnerability type and contextual complexity. Straightforward coding errors were detected with relatively high consistency, whereas vulnerabilities requiring architectural reasoning, authentication-flow analysis, or threat-model awareness were more frequently overlooked. Similar observations have been reported in cybersecurity benchmarking efforts and threat-intelligence evaluation frameworks, which emphasize the importance of domain-specific validation when assessing large language model capabilities (Alam et al., 2024; Bhusal et al., 2024).

DOI: 10.46121/pspc.53.4.35

7.3 Style Drift as a Governance and Maintainability Problem

While hallucination and vulnerability exposure directly affect correctness and security, style drift introduces longer-term challenges associated with software maintainability and organizational governance. The experimental results showed that AI-assisted reviewers occasionally recommended modifications that conflicted with established coding standards, naming conventions, architectural patterns, or documentation practices. Although such deviations may not immediately affect functionality, their cumulative impact can significantly increase maintenance costs and reduce codebase consistency over time.

The prevalence of style drift supports previous research indicating that generative AI systems often optimize for locally plausible solutions rather than organization-specific development standards (Geladin Jr, 2024; Nygård, 2024). This tendency becomes particularly problematic in large software projects where consistency and standardization are essential for long-term maintainability. As AI-generated recommendations become more deeply integrated into development workflows, unmanaged style drift may gradually contribute to technical debt accumulation and architectural fragmentation.

The findings also suggest that style drift extends beyond formatting preferences. In many cases, recommendations altered design patterns, refactoring approaches, and implementation structures in ways that diverged from repository conventions. Similar concerns have been identified in studies examining AI-assisted coding practices and large-scale deployment of generative software engineering tools (Murali et al., 2024; Vijayvergiya et al., 2024). Consequently, organizations should consider style consistency metrics as an important component of AI system evaluation rather than focusing exclusively on defect detection accuracy.

7.4 Implications for Enterprise Deployment

The combined effects of hallucination, vulnerability exposure, and style drift have important implications for enterprise adoption of AI-assisted code review systems. The results indicate that these technologies can provide substantial productivity benefits and support developers in identifying routine defects; however, their outputs cannot yet be treated as independently reliable sources of software assurance. Instead, AI-generated reviews should be considered advisory artifacts subject to verification and governance controls.

Organizations deploying AI-assisted review systems should establish risk-based review policies that determine when human intervention is required. High-impact code changes involving authentication systems, cryptographic functions, financial transactions, infrastructure automation, or safety-critical operations should remain subject to mandatory human validation. Such controls help reduce the probability that hallucinated findings or incomplete security recommendations will propagate into production environments.

The findings also support the need for continuous monitoring and operational evaluation. Large language model performance is influenced by evolving training methodologies, deployment configurations, and organizational usage patterns. As AI technologies become increasingly integrated into software engineering ecosystems and operating-system-level development environments, maintaining consistent assurance standards will require ongoing benchmarking and governance oversight (Zhang et al., 2024).

Additionally, benchmark contamination represents a growing concern for enterprise evaluation programs. If models have previously encountered benchmark content during training, reported performance may overestimate real-world effectiveness. This issue underscores the importance of continuously updating benchmark datasets and maintaining evaluation scenarios that accurately reflect operational software development environments (Xu et al., 2024).

7.5 Relationship to Prior AI Code-Review Benchmarks

The results broadly validate the three-dimensional risk taxonomy proposed for AI-generated code reviews, confirming that hallucination, vulnerability exposure, and style drift represent distinct but interconnected dimensions of AI review quality (Azmi, 2023). Rather than occurring independently, these failure modes frequently interacted during evaluation. For example, hallucinated explanations sometimes led to inappropriate remediation advice, while style-related recommendations occasionally introduced security-relevant modifications. This interdependence suggests that comprehensive evaluation frameworks should assess multiple dimensions simultaneously rather than relying on single-metric performance measures.

DOI: 10.46121/pspc.53.4.35

The benchmark design developed in this study also extends existing large language model assessment methodologies by integrating software engineering, cybersecurity, and maintainability perspectives within a unified evaluation framework. This approach aligns with broader efforts to establish structured risk taxonomies and benchmark-driven assessment methodologies for AI systems (Cui et al., 2024). Similar to hazard-analysis benchmarking in safety-critical domains, the findings demonstrate that accurate performance evaluation requires more than measuring correctness alone; it must also consider the potential consequences of erroneous outputs and the contexts in which they are generated (Dokas, 2024).

Overall, the results indicate that AI-assisted code review systems are valuable contributors to modern software engineering workflows but should not be regarded as substitutes for established assurance mechanisms. Their effectiveness is highest when deployed within controlled governance frameworks that combine automated assistance, human expertise, security validation, and continuous benchmarking. Such an approach enables organizations to benefit from AI-driven productivity improvements while maintaining appropriate levels of reliability, security, and software quality assurance.

GOVERNANCE RECOMMENDATIONS

The findings of this study indicate that AI-assisted code review systems should be governed as software-assurance technologies rather than standalone productivity tools. Although these systems demonstrate substantial value in accelerating review processes and identifying common coding defects, their susceptibility to hallucinated findings, incomplete security reasoning, and style inconsistency necessitates the implementation of structured governance mechanisms. Effective governance must address technical reliability, security assurance, operational accountability, and continuous performance monitoring throughout the software development lifecycle (Azmi, 2023; Cui et al., 2024).

8.1 Human-in-the-Loop Escalation Thresholds

Human oversight remains essential for mitigating risks associated with automated review recommendations. AI-generated review outputs involving security-sensitive components, authentication mechanisms, cryptographic implementations, access-control logic, and infrastructure configurations should be automatically escalated for expert validation before acceptance. Previous studies have demonstrated that while AI systems can effectively identify routine defects, they often struggle with contextual reasoning and nuanced security assessments, increasing the possibility of inaccurate or incomplete recommendations (Murali et al., 2024; Raghavan et al., 2023). Establishing risk-based escalation thresholds ensures that high-impact code modifications receive appropriate human scrutiny while preserving productivity benefits for lower-risk review tasks.

8.2 Provenance and Auditability of AI Review Outputs

Organizations should maintain comprehensive audit trails documenting AI-generated review comments, recommended code modifications, reviewer responses, and final acceptance decisions. Such provenance mechanisms facilitate accountability, support compliance requirements, and enable retrospective analysis of review failures. Transparent documentation of model versions, prompts, review contexts, and generated outputs also assists in identifying sources of hallucination and evaluating model behavior over time (Azmi, 2023; Cui et al., 2024). Auditability is particularly important in regulated environments where software quality and security assurance must be demonstrably verified.

8.3 Pre-Deployment Benchmark Testing

Before integration into production development pipelines, AI-assisted review systems should undergo standardized benchmark evaluations that measure hallucination rates, vulnerability detection performance, vulnerability introduction risk, and style consistency. Benchmark-driven validation provides objective evidence regarding model reliability and operational suitability. Similar benchmark-based assessment approaches have demonstrated effectiveness in evaluating AI performance across cybersecurity, software engineering, and safety-critical domains (Bhusal et al., 2024; Alam et al., 2024; Dokas, 2024). Organizations should establish minimum performance thresholds that models must satisfy before deployment.

8.4 Continuous Drift Monitoring

The performance of AI-assisted review systems may change over time due to model updates, evolving software architectures, changing development practices, and repository growth. Continuous monitoring mechanisms should therefore be implemented to track fluctuations in hallucination frequency, vulnerability detection effectiveness,

DOI: 10.46121/pspc.53.4.35

and style adherence. Periodic reassessment using benchmark datasets can help identify degradation patterns before they negatively affect software quality. Research on hallucination behavior and model evaluation demonstrates that performance stability cannot be assumed across operational environments and must be continuously verified (Nadeau et al., 2024; Tonmoy et al., 2024).

8.5 Secure Integration into DevSecOps Pipelines

AI-assisted code review should be integrated within broader DevSecOps workflows rather than functioning as an isolated review mechanism. Automated review recommendations should be complemented by static analysis tools, dynamic testing frameworks, vulnerability scanners, and human validation processes. Multi-layered verification reduces the likelihood that hallucinated findings or insecure recommendations propagate into production systems. Hybrid approaches combining AI capabilities with traditional security validation mechanisms have shown promise in improving overall assurance outcomes while reducing operational risk (Raghavan et al., 2023; Vijayvergiya et al., 2024). Consequently, AI-assisted review systems should serve as decision-support components within a comprehensive software assurance ecosystem.

LIMITATIONS AND FUTURE RESEARCH

9.1 Limitations

Several limitations should be considered when interpreting the results of this study. First, the benchmark datasets employed in the evaluation, although designed to include diverse vulnerability classes, coding styles, and review scenarios, cannot fully represent the complexity of real-world enterprise software environments. Large-scale industrial systems often contain domain-specific architectures, proprietary frameworks, and unique operational constraints that may influence review performance differently than benchmark repositories (Geladin Jr, 2024; Nygård, 2024).

Second, the study focuses primarily on observable review outputs and does not directly evaluate the internal reasoning processes of the AI systems under investigation. Consequently, it is possible that some correct recommendations were generated through flawed reasoning pathways, while certain incorrect recommendations may have resulted from incomplete contextual information rather than inherent model deficiencies (Poldrack et al., 2023).

Third, benchmark-driven evaluations are susceptible to data contamination risks. If evaluation samples overlap with training data or are represented within publicly available repositories used during model development, performance estimates may be artificially inflated. Such contamination challenges have been identified as a growing concern in the evaluation of large language models and may affect the generalizability of reported results (Xu et al., 2024).

Fourth, the study evaluates AI-assisted review performance under controlled experimental conditions. Actual software development environments involve collaborative workflows, organizational policies, varying developer expertise levels, and evolving project requirements that may influence the effectiveness of AI-generated review recommendations (Murali et al., 2024).

Finally, the rapid evolution of foundation models introduces challenges for longitudinal validity. Improvements in model architecture, fine-tuning strategies, and domain adaptation techniques may alter performance characteristics over time, requiring continuous reassessment of benchmark conclusions (Zhang et al., 2024; Cui et al., 2024).

9.2 Future Research

Future research should explore benchmark frameworks capable of evaluating emerging agent-based software engineering systems that perform autonomous review, testing, and remediation activities.As AI systems gain the ability to tackle multi-step development tasks with increasing efficiency and proficiency, novel evaluation approaches will be needed to ensure the reliability and security consequences of autonomous software engineering workflows (ASEWs) (Zhang et al., 2024).

There is a need for further research on Explainable AI Mechanisms for Code Review. Greater clarity about the "why" of generated recommendations might help reviewers better trust the recommendations and detect errors,

DOI: 10.46121/pspc.53.4.35

and minimize the effects of hallucinated results. There may also be an improvement in enterprise governance and compliance capabilities by using explainability-focused approaches (Tonmoy et al., 2024).

Further research is needed to explore the relationship between AI-powered review systems and secure software development practices. Evaluating the combined effectiveness of AI review, static analysis, dynamic testing and threat-modelling techniques can help bring to light valuable insights on how to build a strong software assurance framework (Raghavan et al., 2023; Bhusal et al., 2024).

Another promising trajectory is going toward the creation of domain-specific benchmarks for specific industries like healthcare, finance, industrial control systems and critical infrastructure. In such settings, reliability and security are crucial, and failure modes might not manifest in the same way as in a standard-purpose software repository (Dokas, 2024).

Further research on the effects of extended adoption of AI tools for review should examine their organizational implications, such as impact on developer knowledge, review culture, coding-standard evolution and technical debt. Longitudinal studies that explore how humans and AI work together would be invaluable in understanding the overall impact of AI in software engineering processes (Murali et al., 2024; Vijayvergiya et al., 2024).

Future benchmarking efforts ought to include comprehensive risk assessment methods that all-in-one assess hallucination, vulnerability exposure, style drift, factual consistency, security robustness, and operational reliability. The development of unified evaluation frameworks would enable a more comprehensive picture of the performance of AI-supported code review, and the ability to establish more trustworthy software assurance systems to comply with the requirements of enterprise and security applications. (Azmi, 2023, Cui et al., 2024, Alam et al., 2024).

CONCLUSION

As AI-powered code review tools become more prevalent, they have ushered in a new era of possibilities for streamlining software development, catching bugs, and enhancing code quality.The rise in popularity of AI-driven code review systems has opened a wealth of potential to boost software development efficiency, bug detection, and code quality. According to the results of this study, however, these systems come with a number of significant issues concerning hallucinated review results, vulnerability exposure and style drift as well. AI reviewers can effectively detect a variety of coding mistakes and typical security issues, but they do not perform as well on complex security situations, unclear code sections, or coding requirements that differ by organization.

In this research, we proposed a benchmark framework that systematically assesses these three critical failure modes in AI-assisted code reviews workflows. The results show that hallucinations can decrease review reliability, missing security reasoning can miss vulnerabilities or unsafe security recommendations, and style drift can impact long-term maintainability. The results underscore the importance of thoroughly evaluating AI review systems prior to their adoption in business and security-critical settings.

The study also shows the value of governance activities like human-in-the-loop validation, benchmark testing, review-output auditability and ongoing performance monitoring. Instead of replacing human skills, AI code review tools should be viewed as one of several tools in software assurance and DevSecOps toolboxes.

In summary, the proposed benchmark provides a practical framework to assess the impact on the trustworthiness, security effectiveness, and maintainability of AI-powered code review tools. Given the ongoing advancement of AI technologies, companies should prioritize validating, monitoring, and managing the risks associated with using automated review tools to improve software quality while keeping new reliability and security issues at bay (Azmi, 2023; Murali et al., 2024; Vijayvergiya et al., 2024; Cui et al., 2024).

REFERENCES

1. Azmi, S. K. (2023). Trust but verify: Benchmarks for hallucination, vulnerability, and style drift in AI-generated code reviews. *Well Testing Journal*, *32*(1), 76-90.

DOI: 10.46121/pspc.53.4.35

2. Geladin Jr, D. M. (2024). *Analyzing the Impact of Generative AI Models on IaC Code Quality* (Doctoral dissertation, National University).
3. Murali, V., Maddila, C., Ahmad, I., Bolin, M., Cheng, D., Ghorbani, N., ... & Rigby, P. C. (2024). Ai-assisted code authoring at scale: Fine-tuning, deploying, and mixed methods evaluation. *Proceedings of the ACM on Software Engineering*, *1*(FSE), 1066-1085.
4. Zhang, Y., Zhao, X., Li, Z., Cheng, G., Yin, J., Zhang, L., & Chen, Z. (2024). Integrating Artificial Intelligence into Operating Systems: A Survey on Techniques, Applications, and Future Directions. *arXiv preprint arXiv:2407.14567*.
5. Nygård, J. (2024). *AI-assisted code generation tools* (Master's thesis, J. Nygård).
6. Vijayvergiya, M., Salawa, M., Budiselić, I., Zheng, D., Lamblin, P., Ivanković, M., ... & Just, R. (2024, July). Ai-assisted assessment of coding practices in modern code review. In *Proceedings of the 1st ACM international conference on AI-powered software* (pp. 85-93).
7. Raghavan, K., Padmanabhan, N., & Venkatesh, A. (2023). AI-Assisted Hybrid Vulnerability Analytics for RESTful APIs: Static Rule Matching, Dynamic Fuzzing, and Exploitability Validation. *Journal of AI Analytics and Applications*, *1*(4), 20-35.
8. Poldrack, R. A., Lu, T., & Beguš, G. (2023). AI-assisted coding: Experiments with GPT-4. *arXiv preprint arXiv:2304.13187*.
9. Dokas, I. M. (2024). From hallucinations to hazards: benchmarking LLMs for hazard analysis in safety-critical systems. *Safety*, 2.
10. Cui, T., Wang, Y., Fu, C., Xiao, Y., Li, S., Deng, X., ... & Li, Q. (2024). Risk taxonomy, mitigation, and assessment benchmarks of large language model systems. *arXiv preprint arXiv:2401.05778*.
11. Nadeau, D., Kroutikov, M., McNeil, K., & Baribeau, S. (2024). Benchmarking llama2, mistral, gemma and gpt for factuality, toxicity, bias and propensity for hallucinations. *arXiv preprint arXiv:2404.09785*.
12. Bhusal, D., Alam, M. T., Nguyen, L., Mahara, A., Lightcap, Z., Frazier, R., ... & Rastogi, N. (2024, December). Secure: Benchmarking large language models for cybersecurity. In *2024 Annual Computer Security Applications Conference (ACSAC)* (pp. 15-30). IEEE.
13. Tonmoy, S. M., Zaman, S. M., Jain, V., Rani, A., Rawte, V., Chadha, A., & Das, A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*.
14. Alam, M. T., Bhusal, D., Nguyen, L., & Rastogi, N. (2024). Ctibench: A benchmark for evaluating llms in cyber threat intelligence. *Advances in Neural Information Processing Systems*, *37*, 50805-50825.
15. Xu, C., Guan, S., Greene, D., & Kechadi, M. (2024). Benchmark data contamination of large language models: A survey. *arXiv preprint arXiv:2406.04244*.

DOI: 10.46121/pspc.53.4.35