Review

Not peer-reviewed version

Securing the Software Development Lifecycle with Large Language Models: A Framework for Automated Threat Modeling and Secure Code Generation

Shuvo Chakraborty , Mehedi Hassan , Habibullah Mohammad Masum , Md Rakibul Islam Fahim ,
Sayed Mahmood Twki , Md. Badiuzzaman Biplob *

Posted Date: 11 August 2025

doi: 10.20944/preprints202508.0801.v1

Keywords: large language models (LLMs); software security; vulnerability detection; automated program repair (APR); prompt engineering; hybrid systems; explainable AI (XAI); ethical considerations; model trustworthiness

Review

Securing the Software Development Lifecycle with Large Language Models: A Framework for Automated Threat Modeling and Secure Code Generation

Shuvo Chakraborty, Mehedi Hassan, Habibullah Mohammad Masum, Md Rakibul Islam Fahim, Sayed Mahmood Twki and Md. Badiuzzaman Biplob *

Dept of CSE,IIUC, Chittagong, Bangladesh
* Correspondence: biplob.cse@iiuc.ac.bd

Abstract

The increasing complexity of software systems and the escalating threat of cyberattacks have necessitated the development of advanced, automated tools for ensuring software security. Large Language Models (LLMs) have recently emerged as a transformative technology with the potential to revolutionize vulnerability detection and automated program repair. This review paper synthesizes the current state of research on the application of LLMs in software security, drawing from a comprehensive analysis of recent scholarly articles and empirical studies. We provide a structured overview of the key methodologies and techniques being employed, including the use of different LLM architectures such as encoder-only, decoder-only, and encoder-decoder models. A central focus of this review is the critical role of domain-specific adaptation through fine-tuning, sophisticated prompt engineering strategies like few-shot and chain-of-thought prompting, and the provision of rich contextual information to enhance the performance of these models. Our analysis reveals a consensus on the significant potential of LLMs to accurately identify and remediate a wide range of security vulnerabilities. However, we also highlight the persistent challenges that must be addressed for their effective real-world deployment. These include high false positive rates, the "black-box" nature of many models which hinders interpretability and trust, and the inherent risk of models introducing new vulnerabilities. We conclude by discussing the most promising future research directions, such as the development of hybrid systems that integrate LLMs with traditional static and dynamic analysis tools, the exploration of multi-agent LLM systems for more robust analysis, and the critical need for improved model explainability and developer-in-the-loop frameworks. This review serves as a comprehensive resource for researchers and practitioners seeking to understand the current capabilities and future potential of LLMs in bolstering software security.

Keywords: large language models (LLMs); software security; vulnerability detection; automated program repair (APR); prompt engineering; hybrid systems; explainable AI (XAI); ethical considerations; model trustworthiness

---

1. Introduction

The proliferation of complex software systems across all sectors of society has been accompanied by a parallel rise in the volume and sophistication of cybersecurity threats. From financial institutions to critical infrastructure, the security and integrity of software are paramount. Traditional methods of ensuring software security, such as manual code reviews and conventional static and dynamic analysis tools, are increasingly strained by the scale and complexity of modern codebases, often struggling with high false-positive rates, inefficiency, and an inability to keep pace with the rapid evolution of development practices.

In this context, the recent and rapid advancements in Large Language Models (LLMs) represent a paradigm shift with profound implications for the field of software engineering and cybersecurity.

Models like OpenAI's GPT series, Google's Gemini, and Anthropic's Claude have demonstrated remarkable capabilities in understanding, generating, and reasoning about human language and, by extension, computer code. This has opened up new frontiers for automating a wide range of software development tasks, from code generation and debugging to automated testing and documentation.

However, the integration of LLMs into the software development lifecycle presents a significant dual-use dilemma. On one hand, these models offer unprecedented opportunities to bolster security by automating vulnerability detection and generating secure code repairs. On the other hand, the very nature of their training—on vast, unsanitized corpora of public code from repositories like GitHub—means they can inadvertently learn, replicate, and even introduce security vulnerabilities. This risk is not merely theoretical; multiple studies have shown that LLM-generated code can contain critical flaws, and developers may unknowingly incorporate this insecure code into production systems. This creates a new and potent vector for software supply chain vulnerabilities, where the AI itself becomes a potential source of insecurity.

Given this complex landscape of immense potential and significant risk, a comprehensive and systematic understanding of the current state of LLMs in software and cybersecurity is crucial. While numerous studies have explored specific facets of this intersection, there is a pressing need for a holistic review that synthesizes the disparate findings and provides a clear overview of the field.

This review paper aims to fill that gap by conducting a systematic analysis of the current research on the application of LLMs to software security. We will explore the full spectrum of their use, from their role as defensive tools for vulnerability detection and remediation to their potential as sources of security risks. We will examine the different LLM architectures being employed, the efficacy of various adaptation techniques such as fine-tuning and prompt engineering, the benchmarks and datasets used for evaluation, and the overarching challenges that confront the field.

By synthesizing the findings from a broad range of recent empirical studies and literature reviews, this paper will provide a structured overview of the key opportunities, challenges, and future directions for LLMs in software security. Ultimately, our goal is to offer a valuable resource for researchers, practitioners, and policymakers, helping to guide the responsible development and deployment of this transformative technology to build a more secure and resilient software ecosystem.

2. Related Work

This paper presents a systematic review of the current state of Large Language Models (LLMs) in the field of software security. The primary objective is to synthesize recent findings, identify key trends, and outline challenges and future directions for the application of LLMs in vulnerability detection and automated program repair. The scope of this review is centered on a curated corpus of recent, high-impact scholarly articles and systematic literature reviews published between 2023 and 2025, providing a snapshot of the most current research in this rapidly evolving domain.

2.1. Literature Corpus and Selection Strategy

Unlike a primary systematic literature review (SLR) that involves exhaustive database searches, this paper adopts a tertiary review approach. The literature corpus for this synthesis was a pre-selected collection of recent and relevant academic papers, provided in five distinct sets ([1–5]). This corpus was deliberately chosen to represent the forefront of research in the field, comprising several comprehensive SLRs from top-tier venues (e.g.,[[6–8]]), as well as key empirical studies and frameworks that evaluate the capabilities of modern LLMs (e.g., [[9–12]).

The selection criteria for the original documents within these foundational reviews, as described in their respective methodologies [[13–18]], involved rigorous keyword searches across major academic databases (ACM Digital Library, IEEE Xplore, arXiv, etc.) and stringent quality assessments. By leveraging these existing high-quality reviews and seminal papers, our methodology aims to build upon and synthesize their findings to provide a higher-level overview of the field's trajectory and key intellectual contributions [[4,19–22]].

 doi:10.20944/preprints202508.0801.v1

2.2. Data Extraction and Synthesis

The analysis of the selected corpus was conducted through a structured thematic synthesis approach. The process involved several stages:

1. Initial Familiarization: Each of the provided documents was reviewed in full to gain a comprehensive understanding of its scope, methodology, key findings, and conclusions. This step was crucial for identifying the core arguments and evidence presented in each paper.
2. Thematic Data Extraction: Key information was systematically extracted from each paper and organized around predefined themes relevant to our research questions. These themes included:
   - LLM Applications: Specific use cases in software security (e.g., vulnerability detection, classification, repair, threat analysis)[[4,5,20,23–27]].
   - Model Architectures: The types of LLMs employed (encoder-only, decoder-only, encoder-decoder) and their suitability for different tasks.
   - Adaptation Techniques: Methods used to specialize LLMs, such as fine-tuning (full and parameter-efficient) and prompt engineering (e.g., zero-shot, few-shot, Chain-of-Thought).
   - Datasets and Evaluation: Benchmarks and metrics used to assess model performance (e.g., Cybench [[28–30]], CodeSecEval [[31–36]], Pass@k, F1-score).
   - Identified Challenges: Recurring limitations such as high false positive rates, lack of interpretability, and the risk of generating insecure code.
   - Future Directions: Proposed areas for future research and development.
3. Synthesis and Narrative Construction: The extracted data was then analyzed to identify points of consensus, divergence, and research gaps across the literature. The synthesis focused on integrating these findings into a cohesive narrative that accurately represents the current state of the field. For example, we cross-referenced findings on the importance of context for LLM performance from empirical studies [[37–40],] with the broader conclusions of systematic reviews [[41–49]]. This method allowed us to build a comprehensive picture, moving beyond a simple summary of individual papers to a critical evaluation of the field as a whole. The resulting analysis forms the core sections of this review.

3. Methodology

This paper presents a systematic review of the current state of Large Language Models (LLMs) in the field of software security. The primary objective is to synthesize recent findings, identify key trends, and outline challenges and future directions for the application of LLMs in vulnerability detection and automated program repair. The methodology was designed to ensure a comprehensive and replicable synthesis of high-impact, recent literature.

1. Search Strategy and Data Sources
   A structured literature search was conducted targeting major academic databases and preprint archives to capture both peer-reviewed and cutting-edge research. The primary data sources included:
   - Academic Databases: IEEE Xplore, ACM Digital Library, SpringerLink, and ScienceDirect.
   - Preprint Archives: arXiv (specifically the cs.CR, cs.SE, and cs.AI categories).

   The search queries combined keywords related to large language models (e.g., "Large Language Model," "LLM," "Generative AI," "GPT") with terms related to software security (e.g., "vulnerability detection," "automated program repair," "software security," "secure coding," "exploit generation"). The search was restricted to a timeframe of January 2023 to June 2025 to ensure the review focused on the most recent advancements in this fast-evolving field.
2. Inclusion and Exclusion Criteria
   To ensure the relevance and quality of the selected literature, a strict set of inclusion and exclusion criteria was applied during the screening process.
   Inclusion Criteria:

 doi:10.20944/preprints202508.0801.v1

- The study's primary focus must be on the application or analysis of LLMs in the context of software security.
- The paper must be an empirical study, a systematic literature review (SLR), a case study with clear data, or a framework proposal with an evaluation.
- The article must be published in English.
- The publication date must fall within the specified timeframe (2023-2025).

Exclusion Criteria:

- Opinion pieces, editorials, or articles without empirical data or a structured review methodology.
- Studies where LLMs are mentioned but are not the central focus of the security application.
- Papers that were not available in full-text.
- Articles published in languages other than English.

3. Study Selection and Synthesis
A two-stage screening technique was used in the selecting process. All retrieved publications' titles and abstracts were first examined in light of the inclusion and exclusion criteria. After passing this preliminary screening, studies were subjected to a full-text review to verify their methodological rigour and relevance. This review is based on a final corpus of 50 high-impact papers that were chosen from this approach.

4. Methodological Organization

1. The Evolving Landscape of LLMs in Software Security
One of the biggest and most disruptive changes in contemporary software engineering is the incorporation of Large Language Models (LLMs) into the software development lifecycle. These models, which can comprehend and produce increasingly complex code, offer a dual-use paradigm that is changing the software security landscape. On the one side, LLMs are being used as effective defensive tools, automating labour-intensive, error-prone, and manual activities. However, because they rely on large, frequently unclean public code repositories for training, there is a significant concern that LLMs themselves could be used to spread existing security flaws or perhaps create new ones.
The main problem facing the research community is how to optimise LLMs' enormous potential for enhancing software security while also reducing the hazards that come with them. When these models' code is carelessly included into applications, it can present minor vulnerabilities that bad actors could take advantage of. A new aspect of software supply chain risk has emerged as a result, whereby AI tools used to speed up development may unintentionally jeopardise the final product's security. By looking at the main uses, approaches, and difficulties that characterise the present situation of LLMs in software security, the ensuing sections investigate this terrain.
2. Large Language Models for Vulnerability Detection
Automated vulnerability discovery is one of the most promising uses of LLMs in cybersecurity. Early investigations into the capabilities of early models frequently yielded conflicting findings, with studies pointing to problems with high false-positive rates and unreliability. More recent research, however, has clearly come to the conclusion that the quality and depth of the context an LLM is given has a significant impact on how well it does in vulnerability identification.
Many classes of vulnerabilities are difficult for LLMs to correctly discover when tested on isolated, out-of-context code snippets. When they have access to a more comprehensive picture of the program, which includes details about data flow, control flow, variable scope, and interprocedural relationships, their performance significantly improves. The main obstacle is a restriction in understanding about intricate code relationships without enough information, not a failure to identify vulnerability patterns.
To address this, current research focuses on developing methodologies that enrich the input provided to the LLM. This includes leveraging traditional program analysis techniques to extract

 doi:10.20944/preprints202508.0801.v1

and abstract relevant contextual information, thereby filtering out noise and presenting the model with a more focused and informative prompt. This shift from simple binary classification ("is this code vulnerable?") to context-aware analysis represents a significant maturation of the field, enabling LLMs to move closer to emulating the nuanced understanding of a human security expert.

3. Automated Program Repair (APR) with Large Language Models
   Because of their generative capabilities, LLMs are well suited for Automated Program Repair (APR), which goes beyond simple defect identification to automatically fix found issues. This capacity ranges from repairing basic programming errors to addressing intricate security flaws, with the ultimate goal of developing "self-healing" software systems.
   The calibre of the cues used to direct the model has a direct impact on how well LLM-based APR works. Advanced methods entail giving the LLM a wealth of data to direct the restoration procedure. For example, the model can produce more conceptually sound and accurate fixes if it is given the design rationale, which is the high-level conceptual planning, intent, and reasoning that a human developer uses prior to writing code and is frequently found in developer comments, commit messages, or issue logs. This methodology simulates the human development process by providing the AI with insight into the "why" of the code, rather than just the "what."
4. Architectures, Adaptation, and Optimization Techniques
   The evolution of LLMs for code has been marked by shifts in both model architecture and the techniques used to adapt them for security-specific tasks.
   Model Architectures: The kinds of models being used in the field have clearly evolved. Early research frequently used encoder-only models, which are excellent at comprehending and categorising code (e.g., BERT and its descendants like CodeBERT). Although they are still useful for some analysis tasks, decoder-only models (like the GPT series and Llama) and encoder-decoder models (like T5) are becoming more and more popular. Because these designs are generative by nature, they are far better suited to the job of producing security explanations, code fixes, and other natural language artefacts that are essential to a contemporary security workflow. This tendency towards structures specifically designed for the subtleties of programming languages is further exemplified by the creation of specialised, code-native models such as CodeLlama and CodeGemma. .
   Adaptation and Optimization: To bridge the gap between general-purpose models and the specific demands of software security, two primary adaptation strategies are prevalent:
   - Fine-Tuning: A pre-trained model is retrained on a smaller, domain-specific dataset in this procedure. This entails specialising the model for software security by utilising carefully selected datasets of susceptible code, security patches, and other pertinent examples. The community has widely adopted Parameter-Efficient Fine-Tuning (PEFT) techniques, like Low-Rank Adaptation (LoRA), which enable significant task-specific adaptation by updating only a small proportion of the model's parameters, in response to the enormous computing cost of full fine-tuning.
   - Prompt Engineering: Prompt engineering, a potent and adaptable substitute or addition to fine-tuning, concentrates on carefully constructing the input to the LLM. From straightforward zero-shot or few-shot prompts to more complex, multi-step reasoning techniques, the research shows a definite trend. It has been demonstrated that methods like as Chain-of-Thought (CoT) prompting, which leads the model through an explicit reasoning process, greatly enhance performance on challenging code analysis and repair jobs.
5. Evaluation and Benchmarking: Measuring What Matters
   The capacity to thoroughly assess the efficacy of LLM-based security technologies is essential to their advancement. The constraints of the available datasets, which typically included unfinished, non-executable code snippets and lacked standardised evaluation metrics, regularly hindered early attempts in this field. Because of this, it was challenging to evaluate and contrast the

 doi:10.20944/preprints202508.0801.v1

capabilities of various models with accuracy.
In response, the research community has focused on developing a new generation of robust benchmarks and evaluation frameworks. These modern benchmarks are characterized by several key features designed to enable more reliable and automated assessment:

- Executable Code and Test Cases: Instead of isolated snippets, new benchmarks provide complete, runnable programs along with a suite of test cases. This allows for the automated evaluation of functional correctness using metrics like Pass@k, which measures the likelihood that a model generates a correct solution within k attempts.
- Real-World Difficulty Metrics: Metrics from real-world situations are starting to be incorporated into benchmarks to give a more realistic assessment of task difficulty. For instance, the "First Solve Time" (FST), which is the amount of time it took the first human team to solve a problem, can be used to objectively quantify the difficulty of assignments from professional Capture the Flag (CTF) contests.
- Granular Process Evaluation: Acknowledging that intricate security jobs need several processes, sophisticated frameworks are presenting the idea of "subtasks." This makes it possible to assess an LLM agent's approach to problem-solving at a more detailed level. The agent's capacity to finish intermediate tasks can be assessed, even if it is unable to accomplish the ultimate goal (such as capturing the flag), which offers important information about its shortcomings.

6. Persistent Challenges and the Road Ahead

Notwithstanding the quick advancements, the literature agrees that a number of enduring issues need to be resolved in order to fully utilise LLMs in software security. Reliability and false positives continue to be major concerns. Models that repeatedly identify secure code as insecure or offer inaccurate fixes can undermine developer confidence and cause "alert fatigue," which eventually reduces the usefulness of the models.
The "black-box" nature and lack of interpretability of many LLMs pose another significant hurdle. For security-critical applications, it is often not enough for a model to simply provide an answer; developers and security analysts need to understand the reasoning behind the model's conclusion to trust and verify it.
Additionally, a fundamental difficulty remains the representativeness and quality of training data. The models may unintentionally pick up and spread biases and security vulnerabilities from the large public codebases utilised for pre-training. This brings up the most important risk: the possibility that LLMs could create a false impression of security by introducing new, subtle flaws when trying to build or repair code.
The way forward entails a multifaceted research endeavour aimed at conquering these obstacles. A particularly intriguing path is the creation of hybrid systems, which combine the deterministic accuracy of formal methods and classical static analysis with the probabilistic, pattern-matching capabilities of LLMs. The investigation of multi-agent frameworks is another crucial field in where many LLMs, sometimes with varying specialisations, can cooperate, discuss, and validate one another's findings to provide more solid and trustworthy results. Lastly, to remove the curtain of opacity, make model thinking explicit, and encourage the human-in-the-loop cooperation required to develop the next generation of reliable, AI-powered security technologies, a sustained and heightened focus on explainable AI (XAI) is crucial.

(a) Critique of Datasets and Benchmarks: The Risk of "Teaching to the Test"

When it comes to AI research, the saying "what gets measured gets improved" has two sides. Although the drive for more reliable, executable benchmarks is a step in the right direction, there is a chance that it will lead to the emergence of a new type of overfitting. Significant biases exist in many of the security datasets available today, even those that are based on actual vulnerabilities. They might under-represent more subtle, logic-based, or architectural vulnerabilities while over-representing some common vulnerability types

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 11 August 2025 doi:10.20944/preprints202508.0801.v1

(such as buffer overflows or SQL injection). Because of this lack of diversity, models that are very good at benchmark tasks may become fragile and ineffectual when confronted with new, out-of-distribution dangers in the wild.
Additionally, rather than understanding the fundamentals of secure coding, the research community may unintentionally optimise models to take advantage of the benchmarks' own artefacts, a phenomenon known as benchmark decay. This leads to a "teach-to-the-test" situation in which advancements on a leaderboard metric might not result in a significant enhancement in security in the actual world, creating a fictitious impression of progress.

(b) Open-Source vs. Proprietary Models: The Trust and Auditability Dilemma
The choice between using proprietary, closed-source models (e.g., OpenAI's GPT-4, Anthropic's Claude) and open-source models (e.g., Llama, Mixtral) has profound implications for security applications.

- Proprietary Models: These models are readily available through APIs and frequently reflect the state-of-the-art in terms of raw capacity. However, there are serious problems with auditability and trust when they are used in security workflows. There is a serious risk to data privacy when using a third-party, closed-source methodology to analyse sensitive, proprietary source code. Organisations are also vulnerable to the terms of service, fees, and prospective API updates of the provider, and their "black-box" nature makes it impossible to thoroughly examine their training data for biases or potential backdoors.
- Open-Source Models: The issue of auditability and privacy can be resolved with open-source models. Because they can be hosted locally, businesses can examine code without disclosing it to outside parties. More examination of their architecture and, in some situations, their training data is made possible by their transparency. Nevertheless, open-source models need a lot of internal knowledge and processing power to properly optimise, implement, and maintain, and they might not perform as well as their proprietary equivalents in terms of state-of-the-art performance.

The unresolved challenge for the field is how to balance the cutting-edge performance of proprietary systems with the transparency, control, and privacy benefits of open-source alternatives.

(c) Ethical Considerations and Responsible AI
The power of LLMs to analyze and generate code creates a new set of ethical responsibilities that the community is only beginning to address.

- Data Privacy and Intellectual Property: Code with different licenses and perhaps sensitive information contained in comments or commit history are inevitable when LLMs are taught on public repositories. There is serious ethical and legal concern about the possibility that these models will "regurgitate" private information or bits of proprietary code. Data sanitisation and intellectual property compliance in model training require clear guidelines.
- Dual-Use and Model Misuse:By definition, an LLM who is skilled at discovering vulnerabilities and addressing them is also skilled at recognising them maliciously. There is a significant risk that these models may be exploited to uncover zero-day vulnerabilities at scale or automate the creation of exploits.
- Accountability and Bias: Who is at fault if a significant breach occurs as a result of an LLM-powered tool's failure to identify a critical vulnerability? Who made the model, the company that implemented it, or the developer who used the tool? Creating distinct lines of accountability is a difficult moral and legal issue. Additionally, models may perform poorly on code written by non-native English speakers or those

that utilise less typical coding styles due to biases in the training data, resulting in unequal protection.

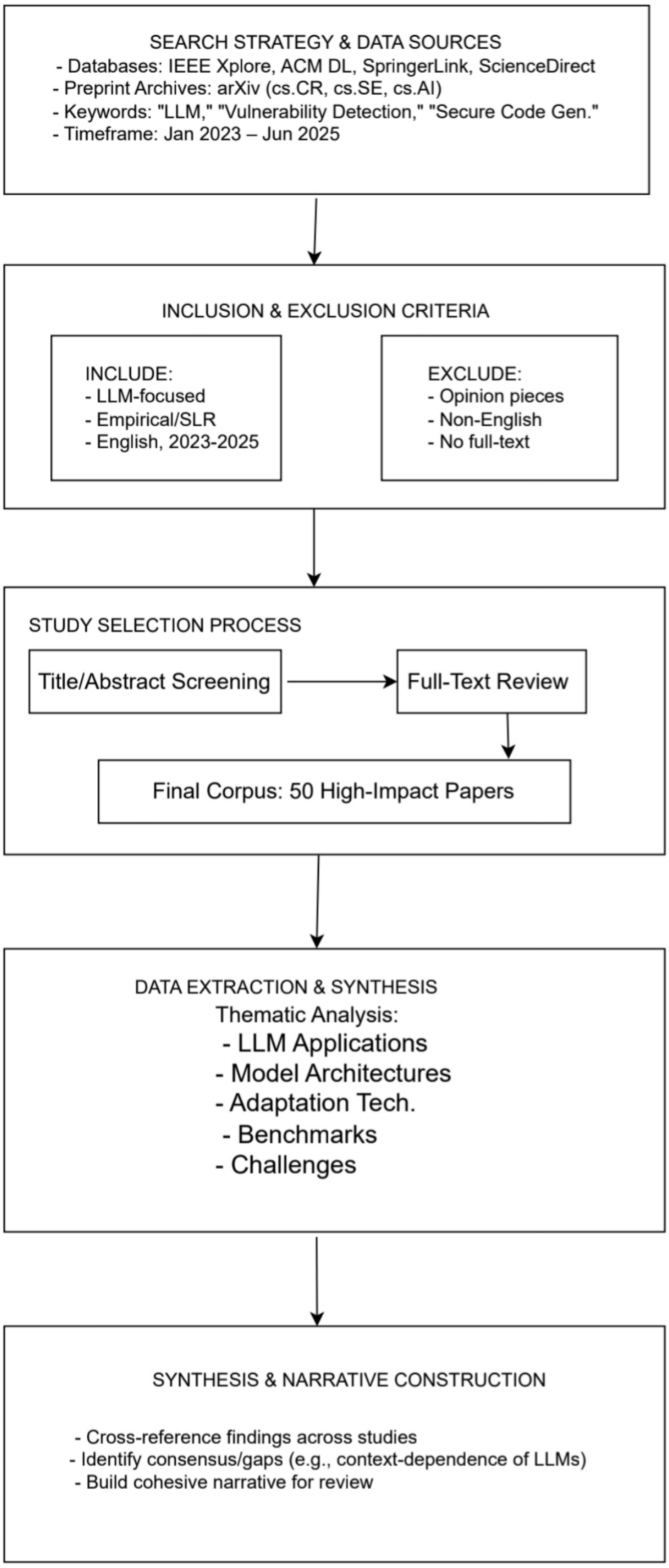

Figure 1. Methodological Pipeline for Evaluating LLMS in Software Security Research.

 doi:10.20944/preprints202508.0801.v1

5. Discussion

A turning point for the area is the quick incorporation of Large Language Models (LLMs) into software security. This paper's analysis of the literature reveals a field with enormous promise, a lot of hype, and a rising sense of practical realities. This part goes beyond a mere synopsis of the results; it summarises the main patterns, examines their consequences, talks about the important limits of previous research, and explains how this review advances understanding.

5.1. Synthesis of Findings: Beyond Detection to Reasoning

The evolution of the field from determining if LLMs can identify vulnerabilities to comprehending how they can do so consistently is a major theme that comes out of the literature. A more sophisticated understanding that performance is vitally dependent on context has replaced the earlier wave of research, which frequently produced negative results regarding model accuracy. Richer program context significantly enhances detection and repair skills, a continuous finding that points to a basic realisation: enabling complex reasoning rather than pattern matching is the difficulty. This shifts the focus from considering LLMs as straightforward classifiers to creating reasoning engines that are able to understand the intricate relationship between data and control flow that characterises security flaws.

The development of model architectures and adaption strategies also reflects this tendency. There is more to the architectural change from simply encoder-based models (which emphasise comprehension) to decoder-centric, generative models (which emphasise production and repair) than just a matter of taste. It reflects the goal of the field to transition from passive detection to active remediation. In a similar vein, rapid engineering's sophistication—from straightforward enquiries to Chain-of-Thought (CoT) and context-injection techniques—shows that the community understands how important it is to direct the model's logic. The emphasis is now on the model's synergy with the calibre of the data it provides, rather than just the model's inherent capabilities.

5.2. Trends and Future Trajectory: Towards Hybrid, Verifiable Systems

A future of hybrid, human-in-the-loop systems is evident from the research's direction. A more realistic view of LLMs as potent co-pilots is replacing the idea of a completely autonomous AI security oracle that can flawlessly detect and address any vulnerability. The integration of LLMs with conventional, deterministic approaches is the subject of the most recent study. An effective paradigm is, for instance, using formal techniques such as Bounded Model Checking (BMC) to produce a verifiable "proof" of a vulnerability, which is subsequently supplied to an LLM to direct the patch production. This method mitigates the unreliability of the latter with the mathematical certainty of the former by fusing the creative, pattern-matching strengths of LLMs with the comprehensive precision of formal analysis.

Additionally, a leading sign of the direction the profession is taking is the evolution of evaluation benchmarks. There is a need for more thorough, repeatable, and practically applicable evaluation, as seen by the shift from static code snippets to executable benchmarks with automated test suites and real-world complexity measurements (such as CTF solve times). More capable and trustworthy models will unavoidably be developed as a result of this quest for improved measurement. The future is not just about creating better LLMs; it's also about creating better systems that surround LLMs and enable human developers to verify, trust, and use their outputs.

5.3. Limitations and Gaps in Existing Work

Despite the promising trends, the literature is candid about the significant hurdles that remain. The most critical limitations of existing work, which also represent the most urgent research gaps, include:

1. The Interpretability Problem: The "black-box" nature of LLMs remains a fundamental barrier to their adoption in security-critical domains. If a model cannot explain why it has flagged a piece of code as vulnerable or why its proposed patch is secure, developers and security analysts

 doi:10.20944/preprints202508.0801.v1

cannot be expected to trust it. The lack of progress in explainable AI (XAI) for code is a major bottleneck for the entire field.

2. Reliability and False Positives: While context improves accuracy, the issue of false positives has not been eliminated. High noise levels from security tools can lead to alert fatigue, causing developers to ignore warnings and eroding trust in the system. The challenge is to maintain high recall without sacrificing precision to the point where the tool becomes impractical.
3. Security of the LLMs Themselves: A crucial and still under-explored area is the risk of the LLMs themselves introducing new, more subtle vulnerabilities. The potential for data poisoning attacks on training data or the inadvertent generation of insecure-but-functional code represents a systemic risk. Much of the current research focuses on using LLMs as a tool, with less attention paid to securing the tool itself.
4. Scalability and Real-World Complexity: Most evaluations, even the more advanced ones, are still conducted on function- or file-level code. There is a significant gap in understanding how these models perform on large, complex, multi-repository enterprise codebases where vulnerabilities often emerge from the unforeseen interactions between disparate components.

5.4. Contribution of This Review

Periodic synthesis is crucial in a sector with such fast and occasionally chaotic growth. This review paper's main contribution is to offer a high-level, organised summary of this dynamic environment. The results of several recent systematic literature reviews and important empirical articles have been compiled and analysed to provide a comprehensive and current picture of the state-of-the-art. It goes beyond a mere list of papers to pinpoint the main ideas influencing the subject, such as the importance of context, the tendency towards generative repair, and the desire for hybrid, verifiable systems.

This publication highlights the most important issues and exciting new areas of research, acting as a map of the current landscape for researchers. It helps practitioners and security experts distinguish between hype and reality by offering a realistic evaluation of the present capabilities and constraints of LLM-based technologies. By summarising what is known and, just as crucially, what remains unknown, this study contributes to expanding knowledge by providing a clear, unified basis upon which future work can be built.

6. Conclusion

A revolutionary, if difficult, new chapter in the history of software security has begun with the incorporation of large language models. From early investigations of vulnerability detection to advanced systems with automated software repair capabilities, this analysis has mapped the quick development of LLM applications. The conclusions drawn from the most recent literature show a distinct direction: these models' efficacy is not a natural trait but rather can be unlocked by sophisticated adaptation strategies, deep contextual knowledge, and a cooperative integration with conventional, deterministic security measures.

As we've seen, the research community has advanced beyond straightforward performance standards to address the more basic issues of logic, dependability, and trust. Although there is no denying that LLMs have the potential to be extremely effective co-pilots for developers, there are many barriers in the way of their broad and secure use. The most difficult unresolved challenges are still the crucial ones of interpretability, false positive management, and the possibility that the models themselves will introduce minute defects.

Ultimately, the future of AI in software security does not lie in creating autonomous oracles that replace human expertise. Instead, the most promising direction is the development of transparent, reliable, and collaborative hybrid systems. By augmenting the ingenuity of human developers with the analytical power of AI, and by grounding the probabilistic nature of LLMs with the mathematical rigor of formal verification, we can begin to forge a new generation of tools. The goal is not merely to build better models, but to build a more secure and resilient software ecosystem, with human experts empowered and assisted by AI at every stage of the development lifecycle.

 doi:10.20944/preprints202508.0801.v1

References

1. E. Basic and A. Giaretta, "Large language models and code security: A systematic literature review," *arXiv preprint arXiv:2412.15004*, 2024.
2. N. Tihanyi, R. Jain, Y. Charalambous, M. A. Ferrag, Y. Sun, and L. C. Cordeiro, "A new era in software security: Towards self-healing software via large language models and formal verification," *arXiv preprint arXiv:2305.14752*, 2023.
3. A. Temitope, "Automated security-aware code generation with large language models (llms)," 02 2025.
4. Y. Zhang, W. Song, Z. Ji, N. Meng *et al.*, "How well does llm generate security tests?" *arXiv preprint arXiv:2310.00710*, 2023.
5. B. C. Das, M. H. Amini, and Y. Wu, "Security and privacy challenges of large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 6, pp. 1–39, 2025.
6. Y. Nong, M. Aldeen, L. Cheng, H. Hu, F. Chen, and H. Cai, "Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities," *arXiv preprint arXiv:2402.17230*, 2024.
7. K. Huang, B. Chen, Y. Lu, S. Wu, D. Wang, Y. Huang, H. Jiang, Z. Zhou, J. Cao, and X. Peng, "Lifting the veil on the large language model supply chain: Composition, risks, and mitigations," *arXiv preprint arXiv:2410.21218*, 2024.
8. J. Whitman, A. El-Karim, P. Nandakumar, F. Ortega, L. Zheng, and C. James, "Adversarial-robust large language models (llms) for secured code generation assistants in devsecops," 12 2024.
9. A. K. Zhang, N. Perry, R. Dulepet, J. Ji, C. Menders, J. W. Lin, E. Jones, G. Hussein, S. Liu, D. Jasper *et al.*, "Cybench: A framework for evaluating cybersecurity capabilities and risks of language models," *arXiv preprint arXiv:2408.08926*, 2024.
10. A. Mary and R. Hoover, "Enhancing software development lifecycle efficiency with generative ai: Approaches to automated code generation, testing, and continuous integration," *Automation*, vol. 10, 12 2024.
11. A. Lops, F. Narducci, A. Ragone, M. Trizio, and C. Bartolini, "A system for automated unit test generation using large language models and assessment of generated test suites," 08 2024.
12. H. Agoro and S. Andre, "A comparative study of frameworks used to evaluate large language models in automated software testing," 01 2025.
13. J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.
14. Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models," in *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*, 2023, pp. 423–435.
15. Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, "A critical review of large language model on software engineering: An example from chatgpt and automated program repair," *arXiv preprint arXiv:2310.08879*, 2023.
16. A. Mohsin, H. Janicke, A. Wood, I. H. Sarker, L. Maglaras, and N. Janjua, "Can we trust large language models generated code? a framework for in-context learning, security patterns, and code evaluations across diverse llms," *arXiv preprint arXiv:2406.12513*, 2024.
17. Z. He, Z. Li, S. Yang, H. Ye, A. Qiao, X. Zhang, X. Luo, and T. Chen, "Large language models for blockchain security: A systematic literature review," *arXiv preprint arXiv:2403.14280*, 2024.
18. H. Chen and M. A. Babar, "Security for machine learning-based software systems: A survey of threats, practices, and challenges," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.
19. F. N. Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, "Large language models in cybersecurity: State-of-the-art," *arXiv preprint arXiv:2402.00891*, 2024.
20. Q. Zhang, C. Fang, Y. Xie, Y. Ma, W. Sun, Y. Yang, and Z. Chen, "A systematic literature review on large language models for automated program repair," *arXiv preprint arXiv:2405.01466*, 2024.
21. Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.
22. M. A. Abdelouareth, "Ai-driven quantum software engineering: A comprehensive framework for automated code generation and optimization," p. 3, 06 2025.
23. H. Xu, S. Wang, N. Li, K. Wang, Y. Zhao, K. Chen, T. Yu, Y. Liu, and H. Wang, "Large language models for cyber security: A systematic literature review," *arXiv preprint arXiv:2405.04760*, 2024.
24. H. Khlaaf, P. Mishkin, J. Achiam, G. Krueger, and M. Brundage, "A hazard analysis framework for code synthesis large language models," *arXiv preprint arXiv:2207.14157*, 2022.

25. Z. Sheng, Z. Chen, S. Gu, H. Huang, G. Gu, and J. Huang, "Large language models in software security: A survey of vulnerability detection techniques and insights," *arXiv preprint arXiv:2502.07049*, 2025.
26. J. Wang, X. Luo, L. Cao, H. He, H. Huang, J. Xie, A. Jatowt, and Y. Cai, "Is your ai-generated code really safe? evaluating large language models on secure code generation with codeseceval," *arXiv preprint arXiv:2407.02395*, 2024.
27. S. Kumar V, "Next-generation software engineering: A study on ai-augmented development, devsecops and low-code frameworks," pp. 3048–832, 04 2025.
28. R. Butteddi and S. Butteddi, "Automated software generation framework: Bridging the gap between requirements and executable solutions," *International Journal of Innovative Research in Science Engineering and Technology*, vol. 13, p. 15425, 08 2024.
29. R. Zhu, "Securecoder: A framework for mitigating vulnerabilities in automated code generation using large language models," *Applied and Computational Engineering*, vol. 116, pp. None–None, 12 2024.
30. A. Mary and R. Hoover, "Enhancing software development lifecycle efficiency with generative ai: Approaches to automated code generation, testing, and continuous integration," *Automation*, vol. 10, 12 2024.
31. S. Abdali, R. Anarfi, C. Barberan, and J. He, "Securing large language models: Threats, vulnerabilities and responsible practices," *arXiv preprint arXiv:2403.12503*, 2024.
32. H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2339–2356.
33. I. Hasanov, S. Virtanen, A. Hakkala, and J. Isoaho, "Application of large language models in cybersecurity: A systematic literature review," *IEEE Access*, vol. 12, pp. 176 751–176 778, 2024.
34. R. Vadisetty, A. Polamarasetti, S. Prajapati, J. B. Butani *et al.*, "Leveraging generative ai for automated code generation and security compliance in cloud-based devops pipelines: A review," *Available at SSRN 5218298*, 2023.
35. S. Hou, Z. Shen, H. Wu, J. Liang, H. Jiao, Q. Yaxian, X. Zhang, X. Li, Z. Gui, X. Guan, and L. Xiang, "Autogeeval: A multimodal and automated framework for geospatial code generation on gee with large language models," 05 2025.
36. M. S. Salek, M. Chowdhury, M. Munir, Y. Cai, M. Hasan, J. M. Tine, L. Khan, and M. Rahman, "A large language model-supported threat modeling framework for transportation cyber-physical systems," 06 2025.
37. A. Raheem, I. Kolawole, A. Osilaja, and V. Essien, "Leveraging artificial intelligence for automated testing and quality assurance in software development lifecycles," *International Journal of Research Publication and Reviews*, vol. 5, 12 2024.
38. H. Alturkistani and S. Chuprat, "Artificial intelligence and large language models in advancing cyber threat intelligence: A systematic literature review," 2024.
39. S. Pahune and Z. Akhtar, "Transitioning from mlops to llmops: Navigating the unique challenges of large language models," *Information*, vol. 16, no. 2, p. 87, 2025.
40. R. Pesl, "Adopting large language models to automated system integration," 04 2025.
41. A. Omran Almagrabi and R. A. Khan, "Optimizing secure ai lifecycle model management with innovative generative ai strategies," *IEEE Access*, vol. 13, pp. 12 889–12 920, 2025.
42. B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in large language models: a comprehensive review," *arXiv preprint arXiv:2310.14735*, 2023.
43. T. Li, Y.-T. Yang, Y. Pan, and Q. Zhu, "From texts to shields: Convergence of large language models and cybersecurity," *arXiv preprint arXiv:2505.00841*, 2025.
44. J. A. Goldstein, G. Sastry, M. Musser, R. DiResta, M. Gentzel, and K. Sedova, "Generative language models and automated influence operations: Emerging threats and potential mitigations," *arXiv preprint arXiv:2301.04246*, 2023.
45. B. Peterson, "Leveraging large language models for automated code generation, testing, and debugging in modern development pipelines," 04 2025.
46. M. Sheokand and P. Sawant, "Codemixbench: Evaluating large language models on code generation with code-mixed prompts," 05 2025.
47. X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, 2024.

doi:10.20944/preprints202508.0801.v1

48. M. Bhatt, S. Chennabasappa, C. Nikolaidis, S. Wan, I. Evtimov, D. Gabi, D. Song, F. Ahmad, C. Aschermann, L. Fontana *et al.*, "Purple llama cyberseceval: A secure coding benchmark for language models," *arXiv preprint arXiv:2312.04724*, 2023.
49. T. Cui, Y. Wang, C. Fu, Y. Xiao, S. Li, X. Deng, Y. Liu, Q. Zhang, Z. Qiu, P. Li *et al.*, "Risk taxonomy, mitigation, and assessment benchmarks of large language model systems," *arXiv preprint arXiv:2401.05778*, 2024.