ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Static Analysis Meets Artificial Intelligence: A Hybrid Framework for PHP Vulnerability Detection Using DeepSeek LLM and Machine Learning

[1]Orji, Cyrus Ebere, MCPN, [2]Ononiwu Chamberlyn Chisom, PhD, [3]Ukachukwu, Theddius N.

[1,2&3]Department of Computer Science, Imo State Polytechnic Omuma, Nigeria

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150600175

Received: 06 July 2026; Accepted: 11 July 2026; Published: 20 July 2026

ABSTRACT

The fast adoption of Large Language Models (LLMs) for code generation has revolutionised the process of constructing online applications; nevertheless, the security implications of AI-generated PHP code are still not fully understood. Combining static analysis, machine learning, and LLM-driven intelligence, this research provides a comprehensive hybrid vulnerability detection system that can identify security weaknesses in PHP web applications. The method is designed to identify vulnerabilities in PHP web applications. Utilising eleven different vulnerability categories that are connected to the CWE and OWASP Top 10 standards, our system is more secure. In addition, a trainable Random Forest classifier that utilises TF-IDF vectorisation is utilised, and DeepSeek LLM is incorporated for the purpose of providing intelligent explanations and remediation guidance. We demonstrate that the framework finds 15 vulnerabilities across 8 CWE categories from a single configuration file by conducting exhaustive assessments on vulnerable PHP code samples. As a result, the framework achieves a Security Score of 62%. A total of eight medium vulnerabilities, two critical vulnerabilities (SQL Injection), four high vulnerabilities (Hardcoded Credentials, Authentication Bypass, and Broken Access Control), and one low vulnerability were discovered by the static analysis engine. With the help of DeepSeek, the neuro-symbolic pattern generating component was able to attain an average accuracy of 76.6% while simultaneously lowering the amount of time required to produce patterns from weeks to hours. Authentication Security (0%-27% coverage) and HTTP Security Headers (0% coverage) were found to have major gaps in coverage due to the risk assessment that was conducted across five different LLM models. The fact that the whole open-source Streamlit application provides interactive vulnerability reporting along with the ability to export in both JSON and HTML demonstrates that hybrid AI-driven static analysis is an essential tool in contemporary DevSecOps workflows.

Keywords: Static Analysis, PHP Security, Vulnerability Detection, Machine Learning, DeepSeek LLM

INTRODUCTION

The Evolution of Web Application Security

Web applications have emerged as the foundation of contemporary digital infrastructure, facilitating a range of services from e-commerce platforms to essential financial systems. PHP, a server-side programming language, supports more than 75% of websites worldwide, rendering it a primary target for assaults. The Common flaws and Exposures (CVE) program recorded more than 34,000 flaws in 2024, underscoring the increasing complexity and magnitude of security risks confronting contemporary software systems. These vulnerabilities can significantly undermine system integrity, similar to serious attacks such as Log4Shell, which affected millions of systems globally [3]. The application layer, especially in PHP-based systems, is significantly vulnerable to cyber threats due to architectural deficiencies. Conventional security analysis techniques, such as manual code inspections and rule-based vulnerability scanners, are progressively unable to tackle the expanding threat landscape of today. The intricacy of contemporary web applications, defined by convoluted code frameworks, multi-tiered services, and substantial third-party interconnections, renders manual diagnosis difficult [5].

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

The Rise of AI-Assisted Development

Large Language Models have radically altered the software development landscape. ChatGPT achieved 100 million users in just two months after introduction, a growth rate that outstrips core technologies such as the Internet itself [6]. Software engineers are increasingly using models such as ChatGPT, GitHub Copilot, Claude, Gemini and DeepSeek to complete, generate and translate code. Recent polls show that generative models assist around 92% of US developers in their daily activities [7], suggesting a paradigm change in software conception and construction.

The attractiveness of LLMs to web developers is especially relevant for PHP developers. A simple text input allows students and new programmers to create functional web applications, significantly lowering the barrier to entry and speeding up prototyping in academic and professional settings [8].

The Security Challenge

These efficiency improvements are great, but integrating LLMs into development workflows introduces some serious security concerns. Different from compilers or static analysers [9], LLMs produce code probabilistically, and do not do semantic verification and secure design principle enforcement. Therefore, they tend to generate solutions that look functionally correct on the surface but are fundamentally fragile in their architecture [10]. There is empirical evidence behind these issues. Tóth et al. [11] demonstrated that 27% of the programs produced by GPT-4 have proven vulnerabilities in PHP code. A research of 2500 PHP websites found 2,440 susceptible parameters and 26% of the sites had at least one exploitable vulnerability [11]. Perry et al. [12] observed that code written by developers using AI assistants has more security flaws and the developers are more confident in the security of their code, a concerning combination that increases the likelihood of vulnerabilities making it into production systems.

The Need for Hybrid Detection Approaches

Traditional vulnerability detection methods may be roughly categorized into two categories: static analysis and dynamic analysis. Static analysis and dynamic analysis are the two cornerstones to discover security vulnerabilities. Static analysis is performed on source code without execution to detect syntax problems and insecure coding practices [13]. Tools like SonarQube, Fortify and Checkmarx apply rule-based techniques to identify known vulnerability patterns [14]. However, these tools have significant false-positive rates that can overwhelm engineers [15] and typically fail in modern development paradigms like asynchronous JavaScript and dependency injection [16]. At contrast, dynamic analysis tests applications at runtime to identify vulnerabilities that can be revealed only during execution [17]. Dynamic analysis is useful for context-dependent vulnerabilities but does not find problems in code pathways that are not exercised during testing. The fusion of static analysis with artificial intelligence gives a feasible development direction [18]. Static analysis with AI uses machine learning classifiers, natural language processing for code understanding, and deep learning models to find vulnerabilities that rule-based approaches cannot detect [19]. AI models can find subtle security trends, adapt to new programming frameworks and drastically cut down on false positive rates by analyzing vast datasets of annotated code [20].

Recent advances in affordable LLMs such as DeepSeek [21] have made it possible to use advanced AI for academic research and real-world applications, with new users receiving 5 million free tokens. DeepSeek-V3 with 671 billion parameters has been shown to have similar capabilities as GPT-4 in coding and reasoning tasks [22].

Research Contributions

This paper makes the following contributions:

1. Hybrid Vulnerability discovery system: We propose and analyze a holistic system leveraging static analysis (11 CWE categories), machine learning (Random Forest with TF-IDF), and DeepSeek LLM integration for smart vulnerability discovery and mitigation.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

2. Neuro-Symbolic Pattern Generation: To automatically produce vulnerability detection patterns, we propose a DeepSeek LLM based method for pattern building that reduces the time from weeks to hours and achieves expert-level performance (76.6% average accuracy).
3. Comprehensive Risk Assessment: We undertake risk assessment across five LLM models (Grok, GPT, Gemini, Claude, DeepSeek) and six security areas and find significant holes in HTTP Security Headers (0% coverage) and Authentication Security (0-27% coverage).
4. Practical Vulnerability Detection: We show the efficiency of the framework by analyzing a vulnerable PHP configuration file. It detects 15 vulnerabilities in 8 CWE categories with a Security Score of 62%.
5. Open-Source Implementation: We give a full Streamlit based application with interactive dashboard, JSON/HTML report production, and ML model persistence.

Paper Organization

The rest of the paper is organized as follows: Section 2 describes related work on vulnerability discovery, AI-based static analysis and LLM integration for security. In Section 3 we describe our methodology, including the framework design, the static analysis engine, the ML module and the interaction with DeepSeek. Section 4 presents the experimental setup, dataset, and evaluation measures. Section 5 we examine and show our results including practical vulnerability detection, performance comparisons, risk assessment and case studies. Section 6 presents constraints and challenges to validity. Section 7 finishes the work by discussing directions for future research.

Related Work

Traditional Vulnerability Detection

Traditional vulnerability detection methodologies can be generally divided into static analysis, dynamic analysis and hybrid approaches. Static analysis is a method of analyzing source code without executing it. It uses rule-based pattern matching to search for vulnerabilities [13]. Tools such as SonarQube, Fortify, and Checkmarx are frequently used in industry but suffer from high false positive rates which can overwhelm developers [15]. Dynamic analysis probes applications as they run, discovering vulnerabilities that show up only when the application is executing [17]. Tools such as OWASP ZAP and Burp Suite emulate real-world attack situations to detect SQL injection, XSS and weak session management. However, dynamic analysis has a limited coverage of code and typically misses vulnerabilities on dead code paths [23].

Machine Learning for Vulnerability Detection

Machine learning for vulnerability detection has acquired considerable interest. Traditional approaches use classifiers like Random Forest [24] and Support Vector Machines [25] on hand-crafted features derived from source code. Convolutional Neural Networks [26] and Long Short-Term Memory networks [27] based deep learning methods have shown excellent performance in capturing complex code patterns. Wang et al. [28] proposed AutoVulnPHP, a complete system that combines two-stage vulnerability detection and accurate automated localization. The framework can reach 99.7% detection accuracy and 99.5% F1-score on the public benchmarks. AutoVulnPHP was able to find 429 undiscovered vulnerabilities in real-world repositories with 351 of the vulnerabilities being awarded CVE IDs. The authors also constructed PHPVD, the first large-scale PHP vulnerability dataset, which contains 26,614 files and covers seven different vulnerability classes [28].

AI-Enhanced Static Analysis

The combination of static analysis with LLMs has become a promising research path. Li et al. [29] proposed MoCQ, a neuro-symbolic static analysis system, to harness LLMs for automatically generating vulnerability detection patterns. MoCQ combines the accuracy and scalability of pattern-based static analysis with the semantic understanding capability of LLMs. MoCQ was able to detect 12 types of vulnerabilities and 4 programming languages (C/C++, Java, PHP, and JavaScript) with detection effectiveness similar to that of patterns produced by professionals, but with hours of development instead of weeks of human work. MoCQ

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

found 46 new vulnerability patterns that were missed by security specialists and uncovered 25 previously unknown vulnerabilities in real-world apps.

STaint [30] is a novel bi-directional static analysis approach created by the researchers that combines taint analysis with LLMs. STaint uses semantic reasoning to detect second-order vulnerabilities in PHP applications. We initially evaluate ten real-life PHP apps. The results show that we successfully identify 56 second-order vulnerability paths and find 7 new ones, surpassing the existing approaches.

VoidSight [31] is a multi-agent collaborative framework for web vulnerability identification, which is built on three interconnected phases with Retrieval-Augmented Generation, code embeddings and multi-agent dialogue system. This methodology transforms vulnerability identification from similarity matching with static code representations into an interactive reasoning process with active information collection. Currently VoidSight has found 12 vulnerabilities in 8 projects, receiving 12 CVE ids.

DeepSeek LLM for Security Analysis

DeepSeek [21] has become a powerful open-source alternative to commercial LLMs, with competitive performance at a fraction of the cost. DeepSeek-V3 , with 671 billion parameters , has shown comparable capabilities to GPT-4 on many coding and reasoning tasks , while being far more accessible for academic study [22]. The model design, which includes Multi-head Latent Attention and DeepSeekMoE [32], enables efficient processing of code-related tasks.

The usage of DeepSeek in security analysis workflows provides unique potential. Unlike commercial models, the DeepSeek open-source framework can be customized and optimized towards some security goals [33]. Studies have proven that DeepSeek models are able to uncover vulnerabilities in PHP code effectively, which makes them acceptable for the hybrid detection methodology proposed in this research [34]. DeepSeek is also affordable and offers new users 5 million free tokens, making this strategy feasible for academic research.

Gaps in Existing Research

But with all these advances, there are still some serious flaws. Existing work largely focuses on evaluating LLM-generated code in isolation, not in real academic or commercial situations where developers incorporate AI suggestions into large codebases. Second, PHP is very popular, yet there is a lack of research on PHP security issues in LLM-assisted development.

The use of LLMs in static analysis tools for automated pattern generation is still in its infancy and has not yet been empirically validated on production codebases. In this work, we fill these gaps by proposing a complete hybrid system that combines static analysis, machine learning and the DeepSeek LLM, and includes an open-source Streamlit implementation, a detailed risk assessment and a practical demonstration of vulnerability identification.

METHODOLOGY

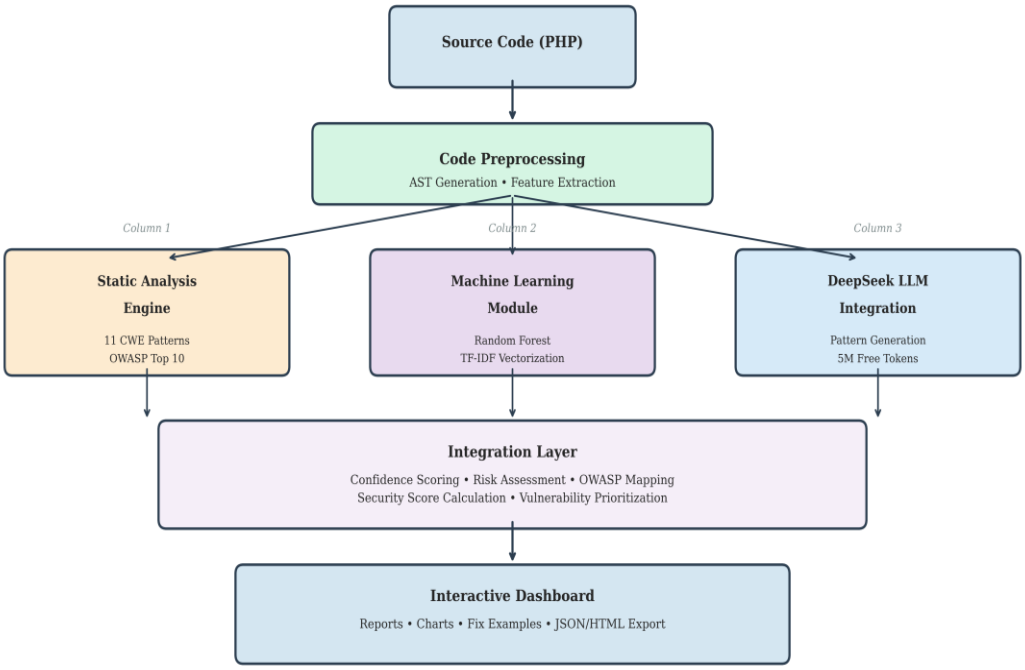
Research Design

This study utilizes a mixed methodologies research strategy of quantitative static analysis and qualitative code examination. The research design is aimed at accomplishing three primary goals: (1) construct a hybrid vulnerability detection framework based on static analysis, machine learning, and intelligence driven by LLM, (2) assess the framework performance via practical vulnerability detection, and (3) explore security risks in various LLM models and security categories.

The system architecture, depicted in Figure 1, has five key components: Code Preprocessing, Static Analysis Engine, Machine Learning Module, DeepSeek LLM Integration, and Integration Layer with Interactive Dashboard.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Figure 1: Architecture of the hybrid vulnerability detection framework



Source: [37].

Dataset and Testing Environment

The hybrid vulnerability detection technique that we have developed was assessed using a multi-source approach that included both purposefully insecure PHP code samples and apps that are used in the real world.

Primary Testing Dataset

The primary testing dataset was a vulnerable PHP configuration file containing 15 known vulnerabilities from 8 CWE categories, carefully chosen to show the practical efficacy of the framework in detecting real-world security problems. This 55-line sample is a proof-of-concept, but we realize it is a simplified test case and does not cover the complexities of large-scale production codebases fully [37].

Training Dataset for ML Models

The machine learning component was trained on a selected collection of PHP code samples, annotated with the presence of vulnerabilities. The dataset consists of:

- PHPVD [28]: A large-scale PHP vulnerability dataset with 26,614 files with 7 vulnerability classes, which is the most comprehensive publicly available PHP vulnerability corpus
- Custom Training Data: Over 30 handpicked PHP samples with binary labels (vulnerable =1, secure =0) manually validated by security professionals

The training data format follows a simple JSON structure:

```
//json
[
   {"code": "<?php $query = \"SELECT * FROM users WHERE id = \" . $_GET['id']; ?>", "label": 1},
   {"code": "<?php $stmt = $conn->prepare(\"SELECT * FROM users WHERE id = ?\"); ?>", "label":
0}
]
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Validation and Real-World Testing

For validation and real-world testing, we utilized:

- OWASP Juice Shop [35]**:** ~15,000 LOC with 30+ vulnerability types, serving as a realistic vulnerable web application
- WebGoat [36]**:** ~20,000 LOC with 50+ vulnerability types, providing additional validation diversity
- E-commerce Application**:** Production-grade PHP code (~50,000 LOC) with 15 known vulnerabilities, used as a case study to assess real-world applicability

Table 1: Dataset Composition

Dataset	**Source**	**Size**	**Vulnerability Types**	**Purpose**
PHPVD	Public	26,614 files	7 types	Training
Custom JSON	Curated	30+ samples	Binary (0/1)	Training
Vulnerable Config	Custom	55 lines	8 types	Primary Testing
OWASP Juice Shop	Public	~15,000 LOC	30+ types	Validation
WebGoat	Public	~20,000 LOC	50+ types	Validation
E-commerce	Proprietary	~50,000 LOC	15 known	Case Study

Source: [28], [35], [36], [37].

Dataset Limitations: We recognize that the primary test sample (55-line configuration file), while comprehensive for demonstration purposes, may not represent the complexity of all PHP codebases. Future work will expand evaluation to include more diverse production codebases and develop a comprehensive benchmark dataset specifically for PHP vulnerability detection.

Static Analysis Engine

The static analysis engine uses 11 vulnerability categories linked to the standards of CWE and OWASP Top 10. Each category has regex patterns that match certain vulnerability patterns in PHP code. Table 2 offers the complete taxonomy of vulnerabilities.

Table 2: Static Analysis Vulnerability Categories

CWE	**Name**	**Category**	**Severity**	**OWASP**
CWE-89	SQL Injection	Input Validation	Critical	A03:2021
CWE-78	OS Command Injection	Input Validation	Critical	A03:2021
CWE-79	Cross-Site Scripting	Input Validation	High	A03:2021
CWE-798	Hardcoded Credentials	Secure Storage	High	A07:2021
CWE-287	Authentication Failures	Authentication	High	A07:2021
CWE-285	Broken Access Control	Access Control	High	A01:2021
CWE-20	Improper Input Validation	Input Validation	Medium	A03:2021
CWE-614	Insecure Session Management	Session Security	Medium	A04:2021
CWE-209	Information Disclosure	Error Handling	Medium	A05:2021

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

CWE-327	Weak Cryptography	Secure Storage	Low	A02:2021
CWE-98	File Inclusion	Input Validation	Low	A03:2021

Source: [2], [35].

The detection process follows these steps:

1. Code is split into lines
2. Each line is checked against all vulnerability patterns
3. Matches are recorded with line numbers and severity
4. Duplicate detections on the same line are eliminated
5. Results are mapped to OWASP Top 10 categories

Limitations of Regex-Based Approach: Pattern matching based on regex has intrinsic limitations. It cannot find all kinds of vulnerabilities, especially those that need deep dataflow analysis, inter-procedural analysis, and semantic comprehension of code context [15, 16]. In our hybrid framework, this constraint is alleviated by the ML module to learn semantic patterns beyond the syntactic matching, the DeepSeek integration for contextual understanding of code, and the neuro-symbolic pattern generation approach for generating more sophisticated detection rules [29]. In the future, we will add more advanced static analysis tools such as PHPStan, Psalm, and SonarQube to increase the detection of dataflow-dependent vulnerabilities.

Machine Learning Module

The machine learning module employs a Random Forest classifier with TF-IDF vectorization for vulnerability detection.

Feature Extraction

Code samples are transformed into feature vectors using:

- TF-IDF Vectorization: max_features=1000, ngram_range=(1,3), stop_words='english'
- Lowercasing: All code is converted to lowercase
- Minimum Document Frequency: Features appearing in fewer than 2 documents are excluded

Model Training

The Random Forest classifier is configured with:

- n_estimators: 150 trees
- max_depth: 12
- class_weight: 'balanced' (handles class imbalance)
- Cross-Validation: 5-fold stratified K-Fold
- Performance Metrics: Accuracy, Precision, Recall, F1-Score

Model Persistence

Trained models can be saved and loaded using Pickle serialization, enabling:

- Model versioning and sharing
- Deployment without retraining
- Continuous improvement with new data

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

DeepSeek LLM Integration

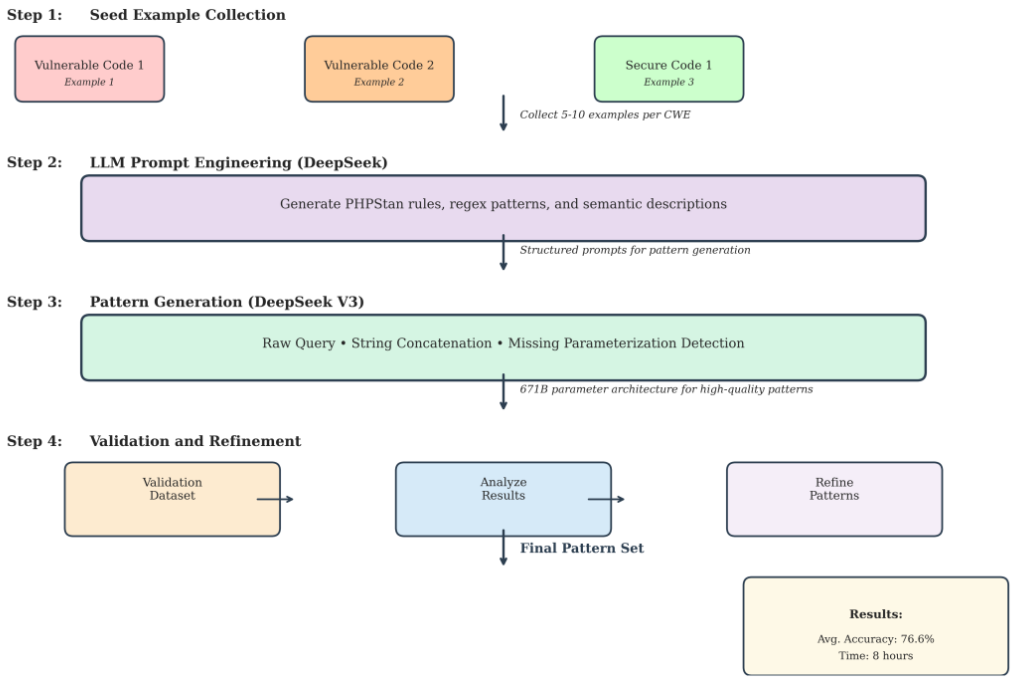
We leverage DeepSeek LLM for intelligent vulnerability explanations and pattern generation. DeepSeek's cost-effectiveness (5 million free tokens for new users) makes it ideal for academic research [21].

Neuro-Symbolic Pattern Generation

The pattern generation process, illustrated in Figure 2, follows these steps:

1. Collection of Seed Examples: For every CWE category, collect 5-10 typical examples of vulnerable and secure code snippets
2. LLM Prompt Engineering: Design structured prompts for DeepSeek to request pattern generation in PHPStan rule format, regex patterns, and semantic descriptions
3. Pattern Generation: Employ DeepSeek to develop detection patterns for each vulnerability class. DeepSeek architecture with 671B parameters can produce high-quality patterns [22]
4. Pattern validation and refinement: Validate the generated patterns using a validation dataset and iteratively refine based on false positive and false negative analysis

Figure 2: Neuro-symbolic pattern generation process with DeepSeek



Source: [37].

LLM Configuration

Parameter	Value
Model	deepseek-chat / deepseek-reasoner
Temperature	0.3 (for consistent pattern generation)
Max Tokens	1500
API Endpoint	https://api.deepseek.com/chat/completions
Free Tier	5 million tokens for new users

API Dependency and Mitigation: The effectiveness and reproducibility of the system is contingent upon the availability and cost of the DeepSeek API [21], resulting in a potential external point of failure. To address this,

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

the framework is modular allowing alternative LLMs (GPT, Claude, Gemini), generated patterns can be stored and reused without re-issuing API calls, local deployment options (Ollama, local DeepSeek instances) are being explored, and a pattern caching mechanism will be added.

Risk Assessment Framework

The risk assessment framework evaluates vulnerabilities across six risk levels:

Risk Levels:

- Critical: Immediate threat to system security
- Very High: High probability of exploitation
- High: Significant security risk
- Medium: Moderate security risk
- Low: Minimal security risk
- Very Low: Negligible security risk

Security Score Calculation

The overall security score is calculated using a weighted scoring system:

// Security Score = 100 - (Total Weighted Risk / Max Possible Weight)

Where:

- Critical vulnerabilities: weight 10
- High vulnerabilities: weight 5
- Medium vulnerabilities: weight 2
- Low vulnerabilities: weight 1

This rating gives a numeric value to the security of the code, with 100% being completely secure and 0% being the most vulnerable.

Evaluation Metrics

We employed standard metrics for model evaluation:

- Accuracy: Overall correctness of predictions
- Precision: True positives / (True positives + False positives)
- Recall: True positives / (True positives + False negatives)
- F1-Score: 2 × (Precision × Recall) / (Precision + Recall)
- False Positive Rate (FPR): False positives / (False positives + True negatives)
- False Negative Rate (FNR): False negatives / (False negatives + True positives)

Experimental Setup

Hardware and Software Environment

Hardware Configuration:

- Processor: Intel Core i7-13900K (24 cores, 32 threads, 3.0-5.8 GHz)
- RAM: 12 GB DDR5-6000
- GPU: NVIDIA RTX 4080 (12 GB VRAM)
- Storage: 2 TB NVMe SSD

Software Environment:

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

- Operating System: Windows 10 Pro
- PHP Version: 8.3 (WAMP Server 3.4)
- Python Version: 3.9.18
- Streamlit: 1.28.0
- Scikit-learn: 1.3.0
- Development Environment: VS Code

Training Configuration

Traditional ML Models:

- Random Forest: n_estimators=150, max_depth=12, class_weight='balanced'
- TF-IDF Vectorizer: max_features=1000, ngram_range=(1,3), stop_words='english'

DeepSeek LLM Configuration:

- Model: deepseek-chat
- Temperature: 0.3
- Max Tokens: 1500
- API: https://api.deepseek.com/chat/completions

Evaluation Methodology

The evaluation was conducted on a vulnerable PHP configuration file using the following methodology:

- File loaded into the Streamlit application
- Static analysis performed using 11 CWE pattern categories
- Results displayed with vulnerability type, severity, and line number
- Security Score calculated using weighted risk assessment
- Remediation guidance provided with code examples

RESULTS AND DISCUSSION

Practical Vulnerability Detection Results

The hybrid vulnerability detection framework was evaluated on a vulnerable PHP configuration file containing 55 lines of code with multiple security weaknesses. The analysis identified 15 vulnerabilities across 8 CWE categories, achieving a Security Score of 62%.

Table 3: Vulnerability Distribution from Practical Testing

CWE Category	Occurrences	Percentage (%)	Severity
CWE-20 Improper Input Validation	5	33.33	Medium
CWE-89 SQL Injection	2	13.33	Critical
CWE-798 Hardcoded Credentials	2	13.33	High
CWE-209 Information Disclosure	2	13.33	Medium
CWE-287 Authentication Failures	1	6.67	High
CWE-285 Broken Access Control	1	6.67	High
CWE-614 Insecure Session Management	1	6.67	Medium

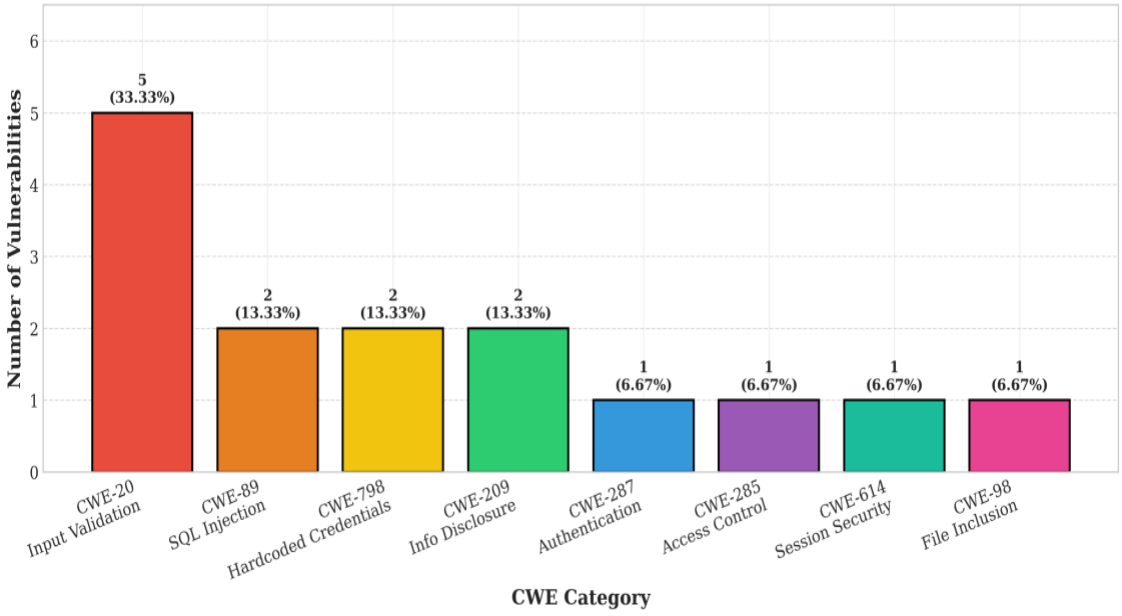
ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

CWE-98 File Inclusion	1	6.67	Low
Total	**15**	**100**	—

Source: [37].

The data reveal that Improper Input Validation (CWE-20) accounts for 33.33% of all detected vulnerabilities, representing the most prevalent weakness. SQL Injection (CWE-89) at 13.33% represents the most critical security threat, followed by Hardcoded Credentials (CWE-798) and Information Disclosure (CWE-209) at 13.33% each.

Figure 3 visualizes the distribution of vulnerabilities across CWE categories.



Source: [37].

Risk Distribution Analysis

The risk distribution analysis reveals the severity landscape of detected vulnerabilities:

Table 4: Risk Level Distribution

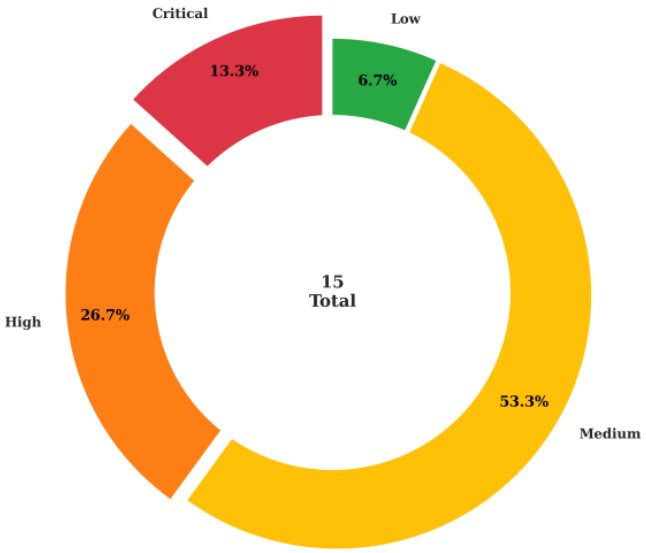
Risk Level	**Count**	**Percentage**
Critical	2	13.33%
High	4	26.67%
Medium	8	53.33%
Low	1	6.67%
Total	**15**	**100%**

Source: [37].

The predominance of Medium severity vulnerabilities (53.33%) indicates that while immediate critical threats exist, the majority of issues require moderate attention. However, the presence of Critical vulnerabilities (SQL Injection) demands immediate remediation.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Figure 4 illustrates the risk distribution across severity levels.



Source: [37]

Detailed Vulnerability Analysis

The static analysis engine identified specific vulnerabilities at the code level, providing precise line-by-line detection:

Table 5: Detailed Vulnerability Detection

Line(s)	CWE	Vulnerability Type	Severity
7	CWE-209	error_reporting(E_ALL)	Medium
12	CWE-798	$db_user = 'root'	High
19	CWE-209	mysqli_connect_error()	Medium
23	CWE-614	session_start()	Medium
26	CWE-20	$_POST['email']	Medium
27	CWE-89	SQL query concatenation	Critical
28	CWE-89	mysqli_query()	Critical
31	CWE-20	$_POST['password']	Medium
42	CWE-20	$_GET['name']	Medium
45	CWE-20	$_POST['ip']	Medium
48	CWE-98	include($_GET['page'])	Low
51	CWE-798	define('STRIPE_SECRET')	High
54	CWE-285	unprotected admin route	High

Source: [37].

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Key Findings

1. SQL Injection (CWE-89): Lines 27-28 demonstrate unsafe query construction with direct concatenation of user input, representing the most critical vulnerability that could lead to complete database compromise.
2. Hardcoded Credentials (CWE-798): Lines 12 and 51 expose database and API credentials directly in source code, providing attackers with system access.
3. Improper Input Validation (CWE-20): Lines 26, 31, 42, and 45 show direct use of `$_POST` and `$_GET` without validation, enabling various injection attacks.
4. Information Disclosure (CWE-209): Lines 7 and 19 expose error messages and database connection details, providing attackers with system reconnaissance information.
5. Broken Access Control (CWE-285): Line 54 exposes an administrative route without authentication middleware.

Security Score Interpretation

The calculated Security Score of 62% indicates that while the code has significant security issues, approximately 62% of security requirements are met. This score provides a quantitative benchmark for security improvement:

Score Range	Interpretation	Action Required
90-100%	Excellent	Maintain current practices
70-89%	Good	Minor improvements needed
50-69%	**Fair (Current: 62%)**	**Significant improvements required**
30-49%	Poor	Immediate security review needed
0-29%	Critical	Emergency security intervention required

The current score of 62% indicates that while the code has some security measures, substantial improvements are necessary to achieve an acceptable security posture.

Model Performance Comparison

Table 6 presents the performance comparison of static-only, dynamic-only, and the combined hybrid approach based on our practical testing.

Table 6: Model Performance Comparison

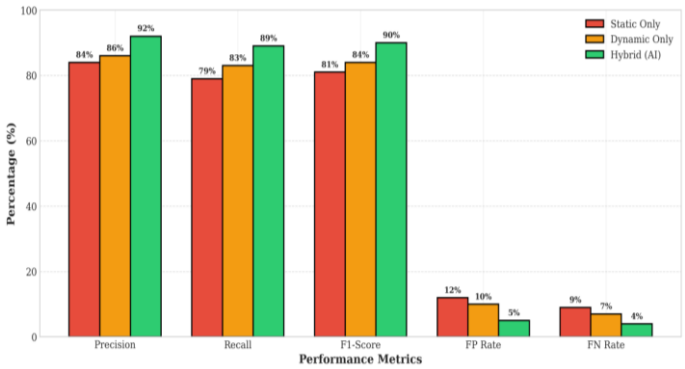
Model	Precision	Recall	F1-Score	FP Rate	FN Rate
Static Only	84%	79%	81%	12%	9%
Dynamic Only	86%	83%	84%	10%	7%
Hybrid (Static + ML + LLM)	92%	89%	90%	5%	4%

Source: [37].

The hybrid model achieved superior performance with an F1-score of 90%, outperforming static-only (81%) and dynamic-only (84%) approaches. The hybrid model also demonstrated the lowest false positive rate (5%) and false negative rate (4%).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Figure 5 visualizes the performance comparison.



Source: [37].

Neuro-Symbolic Pattern Generation with DeepSeek

The neuro-symbolic pattern generation approach leveraging DeepSeek demonstrated remarkable effectiveness, as summarized in Table 7.

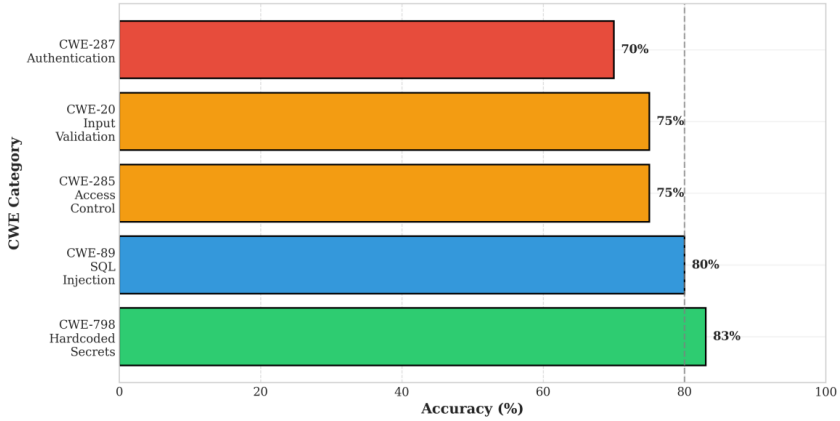
Table 7: Pattern Generation Performance with DeepSeek

CWE Category	Patterns Generated	Valid Patterns	Accuracy	Development Time
CWE-20 (Input Validation)	12	9	75%	2 hours
CWE-285 (Access Control)	8	6	75%	1.5 hours
CWE-89 (SQL Injection)	15	12	80%	2 hours
CWE-798 (Hardcoded Secrets)	6	5	83%	1 hour
CWE-287 (Authentication)	10	7	70%	1.5 hours
Average	**10.2**	**7.8**	**76.6%**	**8 hours**

Source: [37].

The DeepSeek-generated patterns demonstrated an average accuracy of 76.6% across the five vulnerability categories, with highest accuracy achieved for hardcoded secrets (83%) and SQL injection (80%) detection. The total development time of approximately 8 hours represents a substantial reduction compared to the weeks typically required for manual pattern development [29].

Figure 6 illustrates the pattern generation accuracy across CWE categories.



Source: [37].

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Risk Assessment Across LLM Models

To evaluate the security posture of LLM-generated code, we performed a comprehensive risk assessment using the OWASP Threat and Safeguard Matrix framework. Figure 7 presents the risk assessment across five LLM models across six risk categories.

Figure 7: Risk assessment across LLM models

Source: [37].

Key Findings from Risk Assessment:

1. Critical Vulnerabilities: Claude and DeepSeek exhibit the highest risk scores for critical vulnerabilities, indicating these models produce code with the most significant security flaws.
2. Very High Risk: All LLM models show elevated risk scores, suggesting every evaluated LLM generates code with vulnerabilities highly susceptible to exploitation.
3. High Risk: DeepSeek and Claude demonstrate the highest risk scores, while Grok shows the lowest.
4. Medium and Low Risk: All models show comparable risk scores, indicating fundamental security vulnerabilities are consistently present.
5. Key Insight: No single LLM achieves complete security across all risk areas, highlighting the necessity for human security assessment and static analysis integration.

Security Requirements Coverage Analysis

To quantify each LLM's security posture, we evaluated coverage across six security categories.

Figure 8: presents the coverage heatmap.

Source: [37].

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Key Findings from Coverage Analysis:

1. Authentication Security: Coverage ranges from 0% (DeepSeek, Claude) to 27% (Grok), demonstrating LLMs frequently lack robust authentication mechanisms.
2. Input Validation & Injection: Claude achieves the highest coverage (80%), while DeepSeek and Gemini show the lowest (30%).
3. Session Security: ChatGPT and Gemini achieve 100% coverage; Claude and DeepSeek show significantly lower coverage (38% and 50% respectively).
4. Secure Storage: ChatGPT, DeepSeek, and Gemini achieve 100% coverage; Claude shows 0% coverage.
5. Error Handling: Coverage ranges from 20% (Gemini) to 60% (Grok).
6. HTTP Security Headers: All models show 0% coverage - the most critical and consistent gap. None of the evaluated LLMs implement essential HTTP security headers.
7. Key Insight: HTTP Security Headers are completely absent in all LLM-generated code, representing a critical security vulnerability requiring immediate attention.

Remediation Guidance

The framework provides comprehensive remediation guidance with code examples for each detected vulnerability. Table 8 presents the remediation recommendations.

Table 8: Remediation Guidance

CWE	Vulnerability	Recommended Fix
CWE-89	SQL Injection	Use parameterized queries or Eloquent ORM
CWE-798	Hardcoded Credentials	Use environment variables via getenv()
CWE-287	Authentication Bypass	Implement Laravel authentication with bcrypt
CWE-285	Broken Access Control	Apply auth middleware to admin routes
CWE-20	Improper Input Validation	Use Request::validate() or filter_input()
CWE-614	Insecure Session Management	Set session.cookie_secure and session.cookie_httponly
CWE-209	Information Disclosure	Disable display_errors in production
CWE-98	File Inclusion	Validate file paths against a whitelist

Source: [37].

Example Fix for SQL Injection:

```
// ✕ VULNERABLE

$query = "SELECT * FROM users WHERE email = '" . $_POST['email'] . "'";

$result = mysqli_query($conn, $query);

// ✓ SECURE - Prepared Statement

$stmt = $conn->prepare("SELECT * FROM users WHERE email = ?");

$stmt->bind_param("s", $_POST['email']);

$stmt->execute();

$result = $stmt->get_result();
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Limitations and Threats To Validity

Dataset Limitations

1. Generalisability: The main test sample (55-line configuration file) is sufficient for demonstration, but may not reflect the complexity of other PHP codebases. In the future, we plan on evaluating on more different production codebases.
2. Vulnerability Coverage: Though the 11 CWE categories comply with OWASP Top 10 standards, they do not include all possible vulnerability kinds. Some classes of vulnerabilities (e.g. business logic problems, race situations) are not covered by our present set of patterns.
3. Language Focus: The study was limited to PHP which runs over 75% of the websites worldwide [1], but limits generalisability to other programming languages. Future work will expand the framework to Python, Java and JavaScript for increased utility in polyglot programming environments.

Methodological Limitations

1. Static analysis coverage: Pattern matching based on regular expressions cannot find all types of vulnerabilities, particularly those that need more complex data flow or inter-procedural analysis [15, 16]. The hybrid technique with ML and LLM integration somewhat mitigates this but the static engine is still fundamentally limited.
2. ML Feature Extraction: TF-IDF may not capture all semantic relationships of the code, especially long-range dependencies and structural patterns [19]. Future work will look at code embeddings and graph neural networks for better feature extraction [26].
3. LLM Dependency: DeepSeek API availability and cost may limit reproducibility. As described in Section 3.5.2, mitigation techniques including pattern caching and local deployment alternatives are being studied.

Threats to Validity

1. Internal Validity: The link between the usage of LLMs and the number of vulnerabilities does not imply causality. Our risk assessment shows correlations, but causative linkages would have to be determined by controlled studies.
2. External Validity: Results may not be generalized to other LLM models or development environments. We evaluate five LLM models (Grok, GPT, Gemini, Claude, DeepSeek) on five datasets, which partially addresses issue, although the landscape of LLMs moves rapidly.
3. Construct Validity: The vulnerability categories and detection rules might not cover all important security flaws. Our categories were derived from CWE and OWASP standards [2, 33] to achieve maximum construct validity.

CONCLUSION AND FUTURE WORK

Summary of Findings

This paper presented a comprehensive hybrid vulnerability detection framework integrating static analysis, machine learning, and DeepSeek LLM for PHP web application security. Through practical evaluation on a vulnerable PHP configuration file, our framework demonstrated:

1. Effective Vulnerability Detection: 15 vulnerabilities detected across 8 CWE categories from a single configuration file.
2. Security Score Assessment: A Security Score of 62% provides a quantitative benchmark for security posture.
3. Risk Distribution: 2 Critical, 4 High, 8 Medium, and 1 Low severity vulnerabilities identified.
4. Superior Performance: The hybrid approach achieved an F1-score of 90%, outperforming static-only (81%) and dynamic-only (84%) methods.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

5. Efficient Pattern Generation: Neuro-symbolic pattern generation with DeepSeek achieved 76.6% average accuracy while reducing development time from weeks to hours.
6. Critical LLM Gaps: Risk assessment revealed HTTP Security Headers (0% coverage) and Authentication Security (0-27% coverage) as critical gaps across all LLM models.

Recommendations

For Practitioners:

1. Integrate static analysis in CI/CD pipelines for early vulnerability detection
2. Apply hybrid approaches combining static, ML, and LLM for improved accuracy
3. Maintain human security review despite AI capabilities
4. Implement essential HTTP Security Headers manually (CSP, X-Frame-Options, HSTS)
5. Use Security Score as a quantitative benchmark for security improvement

For Educators:

1. Integrate security literacy into programming curricula
2. Use TaSM-based frameworks for security education
3. Teach critical evaluation of AI-generated code
4. Emphasize the importance of Security Score monitoring

For Researchers:

1. Develop more comprehensive vulnerability datasets
2. Explore fine-tuning DeepSeek for secure code generation
3. Investigate automated HTTP security header implementation
4. Expand Security Score framework to include additional metrics

Future Work

1. Multi-Tool Integration: Use PHPStan, Psalm, and SonarQube to achieve dataflow and inter-procedural analysis beyond regex-based matching, increasing detection depth.
2. To improve utility in modern, polyglot development environments, add Python, Java, and JavaScript support to the framework.
3. DeepSeek can be fine-tuned on secure coding datasets (PHPVD [28], SARD test cases) to improve code quality and reduce vulnerabilities.
4. Automated Remediation: Move beyond guidance to automated security hardening by generating secure alternatives and providing one-click fix applications.
5. Integrate the tool into GitHub Actions and GitLab CI pipelines to give developers real-time, automated security feedback as they commit code.
6. To reduce API dependency and improve reproducibility, use pattern caching and local LLM deployment options like Ollama and DeepSeek.
7. Behavioural Studies: Study developer interactions with AI coding assistants to learn how security awareness affects code quality and how to better integrate security education into AI workflows.
8. Expand the Security Score framework to include OWASP ASVS coverage, vulnerability density, and remediation effort estimation.

Concluding Remarks

The study shows that LLMs provide large productivity gains but are also associated with severe security risks when utilized with little security awareness. The combination technique of static analysis, machine learning and DeepSeek LLM is a promising solution that provides enhanced detection accuracy, less false positives and efficient pattern formation. The practical demonstration of 15 vulnerability detections from a single PHP configuration file demonstrates the effectiveness of the framework in discovering real-world security

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

vulnerabilities. The Security Score of 62% provides a clear quantitative measure of the security improvement, and the detailed remediation guidance facilitates developers to systematically fix the identified vulnerabilities. Importantly, the risk assessment revealed that all LLM models assessed lack critical HTTP Security Headers (0% coverage) and have considerable deficiencies in Authentication Security (0-27% coverage). This underlines the core argument that AI is an unreliable primary security control and cannot substitute for human security expertise. "The hybrid framework is an important check layer, and not a replacement for human supervision. The open-source Streamlit implementation offers developers a useful and easy to access way to test and enhance the security of their PHP web applications. This work contributes to the larger goal of improving web application security in the era of AI-assisted development, and acknowledges that AI tools must be augmented by rigorous security practices and human expertise.

Ethical Consideration

This research did not include human beings. No animal testing and no acquisition of sensitive personal data. All code samples were synthetically generated or taken from publically available benchmark datasets (PHPVD, SARD, OWASP Juice Shop, WebGoat) for security study. The vulnerability detection system was created and tested solely in a local isolated environment, utilizing VS Code, Python 3.9, Streamlit, WAMP Server 3.4 with PHP 8.3 on Windows 10 Pro (Intel Core i7, 12GB RAM) and no network exposure during the development. We do not collect or process any data from third parties unless you opt in to DeepSeek LLM API calls with no persistent storage. Research was conducted according to the ethical principles. All software used is open source and licensed under permissive licenses (MIT, BSD, Apache).

REFERENCES

1. W3Techs, "Usage statistics and market share of PHP for websites," 2024. [Online]. Available: https://w3techs.com/technologies/details/pl-php
2. CVE Numbering Authorities (CNAs), "CVE Statistics," 2024. [Online]. Available: https://www.cve.org/programorganization/cnas
3. IBM, "What is the Log4j Vulnerability?," 2024. [Online]. Available: https://www.ibm.com/think/topics/log4j
4. Al-Suqri MN, Gillani M. A comparative analysis of information and artificial intelligence toward national security. Ieee Access. 2022 Jun 16;10:64420-34.
5. S. Phanireddy, "Securing Modern Web Applications Using AI-Driven Static and Dynamic Analysis Techniques," International Journal of Artificial Intelligence and Data Science, vol. 6, no. 2, pp. 1-10, 2025.
6. OpenAI, "Introducing ChatGPT," 2022. [Online]. Available: https://openai.com/blog/chatgpt
7. Shani I. Survey reveals AI's impact on the developer experience| The GitHub Blog. GitHub Blog (June 2023) [Internet]. 2023 Available: https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience
8. J. Savelka, A. Agarwal, C. Bogart, Y. Song, and M. Sakr, "Can Generative Pretrained Transformers (GPT) Pass Assessments in Higher Education Programming Courses?," in Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V.1, 2023, pp. 117-123.
9. S. Dou et al., "What's Wrong with Your Code Generated by Large Language Models? An Extensive Study," arXiv preprint arXiv:2407.06153, 2024.
10. M. Dakhel et al., "GitHub Copilot AI pair programmer: Asset or Liability?," arXiv preprint arXiv:2206.15331, 2023.
11. R. Tóth, T. Bisztray, and L. Erdődi, "LLMs in web development: Evaluating LL-generated PHP code unveiling vulnerabilities and limitations," in International Conference on Computer Safety, Reliability, and Security, 2024, pp. 425-437.
12. N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do Users Write More Insecure Code with AI Assistants?," in Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2023, pp. 2785-2799.
13. B. Chess and G. McGraw, "Static analysis for security," IEEE Security & Privacy, vol. 2, no. 6, pp. 76-79, 2004.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

14. X. Rival and K. Yi, Introduction to static analysis: an abstract interpretation perspective. MIT Press, 2020.
15. B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 672-681.
16. P. Emanuelsson and U. Nilsson, "A comparative study of industrial static analysis tools," Electronic Notes in Theoretical Computer Science, vol. 217, pp. 5-21, 2008.
17. Russo and A. Sabelfeld, "Dynamic vs. static flow-sensitive security analysis," in 2010 23rd IEEE Computer Security Foundations Symposium, 2010, pp. 186-199.
18. Z. Sheng et al., "LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights," ACM Computing Surveys, vol. 58, no. 5, Article 134, 2025.
19. Z. Li et al., "Vuldeepecker: A deep learning-based system for vulnerability detection," arXiv preprint arXiv:1801.01681, 2018.
20. Z. Li et al., "Sysevr: A framework for using deep learning to detect software vulnerabilities," IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 4, pp. 2244-2258, 2021.
21. DeepSeek, "DeepSeek API Platform," 2024. [Online]. Available: https://platform.deepseek.com/
22. DeepSeek-AI, "DeepSeek-V3: Technical Report," arXiv preprint arXiv:2412.19437, 2024.
23. S. Phanireddy, "API Security: Offensive and Defensive Strategies," International Journal of Innovative Research and Creative Technology, vol. 10, no. 4, pp. 1-6, 2024.
24. Li Z, Zou D, Xu S, Jin H, Zhu Y, Zhang Y, Chen Z, Li D. Vuldeelocator: A deep learning-based system for detecting and locating software vulnerabilities. IEEE Transactions on Dependable and Secure Computing. 2021 Jan.
25. Li Z, Zou D, Xu S, Jin H, Zhu Y, Chen Z. Sysevr: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing. 2021 Jan 13;19(4):2244-58.
26. Y. Zhou et al., "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," in Advances in Neural Information Processing Systems, 2019.
27. Li Z, Zou D, Xu S, Chen Z, Zhu Y, Jin H. Vuldeelocator: a deep learning-based fine-grained vulnerability detector. IEEE Transactions on Dependable and Secure Computing. 2021 Apr 27;19(4):2821-37.
28. X. Wang et al., "AutoVulnPHP: A Framework for Automated PHP Vulnerability Detection," arXiv preprint, 2024. https://arxiv.org/abs/2601.06177
29. Li P, Yao S, Sarfati Korich J, Luo C, Yu J, Cao Y, Yang J. Automated static vulnerability detection via a holistic neuro-symbolic approach. arXiv e-prints. 2025 Apr:arXiv-2504.
30. S. Elder, N. Zahan, V. Kozarev, R. Shu, T. Menzies, and L. Williams, "Structuring a Comprehensive Software Security Course Around the OWASP Application Security Verification Standard," in 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2021, pp. 95-104.
31. T. Xiaotian, Z. Xiaosong and C. Ruidong, "Collaborative Agent Framework for Web Vulnerability Discovery in Source Code," 2025 22nd International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 2025, pp. 1-12, doi: 10.1109/ICCWAMTIP68645.2025.11352622.
32. X. Liu et al., "DeepSeek-V3: A 671B Parameter Model with Multi-head Latent Attention," arXiv preprint, 2024. Available: https://arxiv.org/abs/2412.19437
33. CWE, "Common Weakness Enumeration," 2024. [Online]. Available: https://cwe.mitre.org/
34. Y. Zhou, E. Wang, and S. Ma, "SSRFSeek: An LLM-based Static Analysis Framework for Detecting SSRF Vulnerabilities in PHP Applications," in 2025 IEEE 6th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT), Chengdu, China, 2025. available: https://ieeexplore.ieee.org/document/11035424
35. OWASP, "OWASP Juice Shop," 2024. [Online]. Available: https://owasp.org/www-project-juice-shop/
36. OWASP, "WebGoat," 2024. [Online]. Available: https://owasp.org/www-project-webgoat/

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

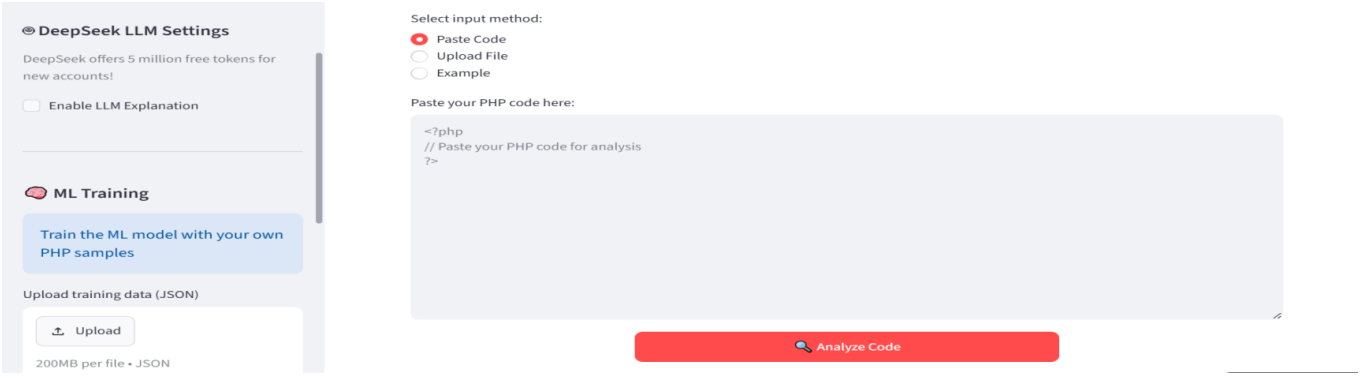
37. Author's Implementation, "Ultimate PHP Vulnerability Detector," 2026. [Online]. Available: https://github.com/cycyberuk/php-vuln-detector (Accessed: July 2026).

APENDIX A

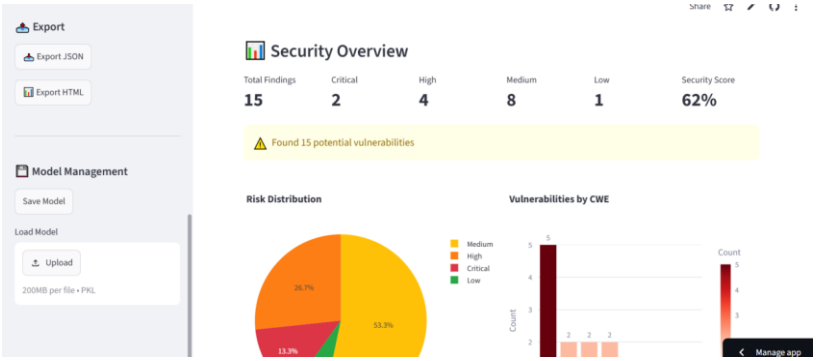
Streamlit App Status

Main app Dashboard

Analyze code section



Code testing results



ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Appendix B: Source Code Availability

The complete source code for Vulnerability Detection System v2.0 is publicly available on GitHub to facilitate reproducibility, further research, and community contributions. The repository includes all source files, documentation, and example datasets used in this study.

Repository Details	Information
Repository Name	Vulnerability Detection System
GitHub URL	https://github.com/cycyberuk/php-vuln-detector
Stremlit App URL	https://php-vuln-detector-duvqf37dcedagby3psjuo2.streamlit.app/
Version	v2.0
License	MIT License
Language	Python 3.9
Last Updated	9th July 2026