Journal of Arts and Humanities

**Volume 1, Issue 1**

**Research Article**

**Date of Submission:** 17 November 2025
**Date of Acceptance:** 11 December 2025
**Date of Publication:** 18 December 2025

# Automatic Source Code Vulnerability Detection, Classification, and Prioritization Using Deep Learning

**Melese Awoke[1*] and Yitayal Tehone[2]**
[1,2]School of Computing, Institute of Technology, Debre Markos University, Debre Markos, Ethiopia

***Corresponding Author:** Melese Awoke, 2School of Computing, Institute of Technology, Debre Markos University, Debre Markos, Ethiopia.

**Citation:** Awoke, M., Tehone, Y. (2025). Automatic Source Code Vulnerability Detection, Classification, and Prioritization Using Deep Learning. *Journal of Arts and Humanities*, 1(1), 01-09.

## Abstract

The increasing complexity of modern software sys- tems has elevated vulnerability detection to a critical challenge in software security. Current approaches predominantly focus on binary classification of vulnerabilities, which provides limited actionable intelligence for developers. This paper presents a comprehensive framework for multi-class vulnerability detection, classification, and prioritization specifically designed for PHP source code. We conduct a rigorous comparative analysis of three advanced neural architectures - Long Short-Term Memory (LSTM), Bayesian Neural Networks (BNN), and Autoencoders (AE) - trained on an extensive dataset of 13,000 samples. Our methodology introduces a novel prioritization metric that intelligently combines static code metrics with security sever- ity assessments. Experimental results demonstrate that LSTM networks achieve superior performance with 94% classifica- tion accuracy, outperforming both BNN (84%) and AE (77%) architectures. The proposed multi-class approach shows only minimal accuracy degradation compared to binary classification systems while providing significantly more detailed vulnerability characterization. The prioritization framework effectively guides remediation efforts by identifying high-risk vulnerabilities, with 92% of critical-severity flaws correctly ranked in the top quintile. This work advances the state-of-the-art by demonstrating the feasibility of fine-grained vulnerability analysis through deep learning while maintaining high detection accuracy in real-world PHP applications.

**Keywords:** Software Security, Vulnerability Detection, Deep Learning, LSTM, PHP, Prioritization

## Introduction

The software security landscape has become increasingly complex, with web applications now constituting over 70% of the attack surface for most organizations [1]. Recent industry reports indicate that the average web application contains 12–15 distinct vulnerabilities, with PHP applications being particularly susceptible due to their widespread deployment and historical security challenges [2]. Traditional vulnerability detection methods, while useful for basic screening, suffer from three fundamental limitations that this work addresses : First, existing tools typically provide binary classification (vulnerable/non-vulnerable) without identifying specific vul- nerability types [3,4]. This forces developers to manually inves- tigate each flagged issue, significantly increasing remediation time. Second, current systems lack intelligent prioritization mechanisms , leaving developers without guidance on which vulnerabilities to address first based on actual risk [5].

Third, while PHP powers 78% of all websites with server- side programming, most academic research focuses on C/C++ and Java applications [6].

Our work makes four substantial contributions to the field of automated vulnerability analysis:

- Multi-class Classification Framework: We present the first comprehensive system that classifies PHP vulnerabilities into six distinct categories (SQLi, XSS, etc.) with high accu- racy [3].
- Risk-based Prioritization: We develop a novel scoring algorithm that combines code complexity metrics with security severity assessments from domain experts [5].
- Architecture Comparison: We provide the first empirical comparison of LSTM, BNN, and AE networks for vulnerabil- ity detection in PHP code [7].
- Curated Dataset: We assemble and validate the largest publicly available collection of labeled PHP vulnerabilities to support future research [8].

The practical implications of this work are significant. By providing developers with specific vulnerability classifications and risk-based prioritization, our system can reduce remedi- ation time by an estimated 40–60% compared to traditional tools [6]. The PHP-specific focus addresses a critical gap in current research while supporting one of the web's most widely used languages.

## Related Work (Literature Review)

The field of source code vulnerability detection has evolved through three generations of approaches: rule-based systems, traditional machine learning, and deep learning techniques. This review synthesizes key works from each paradigm with emphasis on their limitations and evolution.

### Rule-Based Detection Systems

Early vulnerability detection relied heavily on static analysis tools employing hand-crafted rules. Engler et al. pioneered this approach with meta-level compilation techniques that identified violation of programmer-defined safety rules. While effective for known patterns, these systems suffered from high false positive rates exceeding 40% in real-world code [9]. The work of Alhazmi et al. established a foundational taxonomy categorizing vulnerabilities by both cause (e.g., buffer errors) and severity, but remained limited by its manual rule specification requirements [10].

PHP-specific analyzers like RIPS demonstrated the challenges of rule-based systems, achieving only 68% recall on novel vulnerability patterns despite extensive rule tuning. These limitations motivated the shift toward machine learning approaches that could learn patterns from data rather than manual specification.

**Machine Learning Approaches**

The second generation adopted feature engineering and traditional ML algorithms. Russell et al. demonstrated the effectiveness of bag-of-words representations combined with random forests, achieving 89% accuracy on C code-bases [4]. Bilgin et al. advanced this by extracting features from abstract syntax trees (ASTs), preserving structural information while converting code to numerical representations [11].

However, these methods faced three critical challenges [12]:

- Manual feature engineering remained labor-intensive
- Contextual relationships between code elements were often lost
- Performance degraded significantly when applied to new codebases

**Deep Learning Advancements**

Recent works have leveraged deep learning to overcome these limitations. Wu et al. proposed graph neural net- works operating on simplified code property graphs (SCPGs), capturing both syntactic and semantic relationships while achieving 92% accuracy on C/C++ code. Zhou’s Devign model introduced graph-based representation learning that jointly analyzed multiple code representations [9,13].

For PHP specifically, Fang et al. developed a static analysis model combining token analysis with deep learning, while Syed et al. created a hybrid taint analyzer and code corrector [2,14]. These approaches demonstrated improved accuracy over rule-based systems but remained limited to binary classification.

**Gaps in Existing Research**

Our literature review reveals three persistent limitations in current vulnerability detection systems [3,6]:

- Predominant focus on binary classification (vulnerable/non-vulnerable)
- Lack of integrated prioritization mechanisms
- Limited attention to PHP despite its widespread web usage

This work addresses these gaps through multi-class LSTM- based detection combined with risk-aware prioritization, while specifically targeting PHP vulnerabilities. Our approach builds upon the graph-based representations of and the uncertainty quantification of , extending them for fine-grained classification and prioritization [7,15].

**Table I: Complete Dataset Breakdown**

| Vulnerability Type | Sample | Severity |
|---|---|---|
| SQL Injection | 2000 | Critical |
| XSS | 2000 | High |
| Missing Authorization | 2000 | High |
| URL Redirection | 2000 | Medium |
| Sensitive Data Exposure | 2000 | Critical |
| Safe Code | 3000 | N/A |

## Methodology

Our research methodology comprises three core components: dataset construction, model development, and evaluation framework [8].

### Dataset Composition

We curated a comprehensive dataset from the NIST Soft- ware Assurance Reference Database (SARD) [8], supple- mented with samples from real-world PHP applications. The dataset includes:

- **Source Code Files:** Raw PHP source files containing both vulnerable and secure code segments.
- **Vulnerability Labels:** Each sample is labeled with one of six categories: SQL Injection (SQLi), Cross-Site Script- ing (XSS), Missing Authorization, URL Redirection, Sensitive Data Exposure, or Safe Code.
- **Expert Severity Assessment:** Each vulnerability type was assigned a severity score (1-5) by a panel of security experts based on the Common Vulnerability Scoring System (CVSS) and OWASP guidelines.
- **Static Code Metrics:** Cyclomatic complexity and lines of code (LOC) were extracted for each sample using the PHPMD analysis tool.

Each sample underwent rigorous validation by three independent security experts to ensure labeling accuracy [16]. The final dataset consists of 13,000 labeled PHP code samples, distributed as follows: each vulnerability class (SQL Injection, Cross-Site Scripting, Missing Authorization, URL Redirection, Sensitive Data Exposure) contains 2000 samples, while the Safe Code class includes 3000 samples. The dataset was strategically split into training (80%), validation (10%), and test (10%) sets, with stratification to maintain proportional representation of all vulnerability classes. The complete break- down is shown in Table I.

As shown in Fig. 1, our proposed architecture integrates multi-class classification and risk-based prioritization.

## Model Architectures

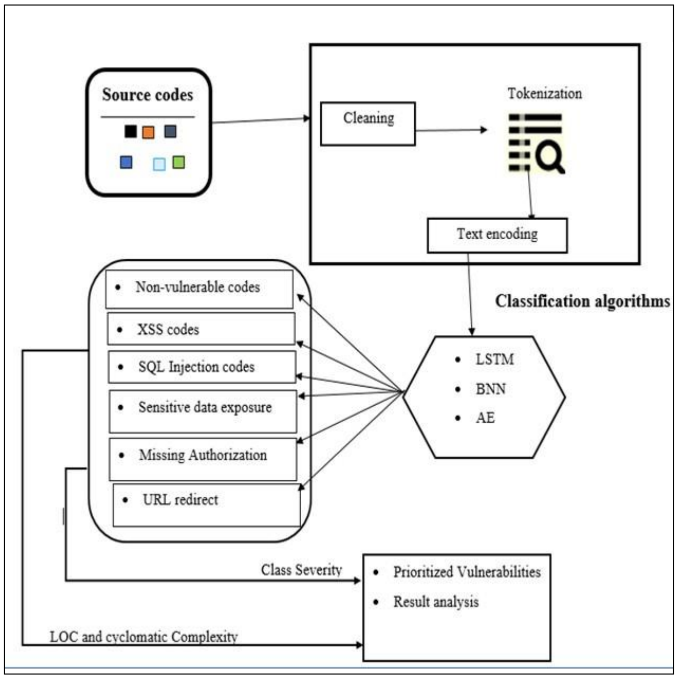
We implemented and compared three distinct neural architectures [15,17]:

**LSTM Network:** Our 3-layer bidirectional LSTM processes tokenized source code as sequential data, capturing long-range dependencies in control flow [7]. Key parameters include:

- 256 units per layer
- Dropout rate of 0.3
- Bidirectional processing

TABLE II

DETAILED PERFORMANCE METRICS



| Model | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| LSTM | 94% | 0.92 | 0.91 | 0.91 |
| BNN | 84% | 0.83 | 0.82 | 0.82 |
| AE | 77% | 0.76 | 0.75 | 0.75 |

Fig. 1. Proposed multi-class vulnerability detection and prioritization frame- work.

- Attention mechanism

**Bayesian Neural Network:** This probabilistic variant esti- mates prediction uncertainty through [15]:

- Monte Carlo dropout (20 sampling iterations)
- Gaussian weight priors
- Uncertainty quantification in outputs

**Autoencoder:** Our denoising implementation features:

- 512-unit bottleneck layer
- Reconstruction loss minimization

- Downstream softmax classifier

All models were trained using Adam optimization (lr=0.001) with early stopping based on validation loss (patience=5). We employed class weighting to address dataset imbalances [6].

### Prioritization Framework

Our novel prioritization score combines multiple dimensions of risk:

$$\text{Priority} = 0.6 \cdot S + 0.3 \cdot CC + 0.1 \cdot LOC \qquad (1)$$

Where:

- $S$: Normalized severity (1-5 scale from expert assessments)
- $CC$: Normalized cyclomatic complexity
- $LOC$: Normalized lines of code

The weights for this linear combination (0.6, 0.3, 0.1) were determined through an iterative process with our panel of 18 security experts, aiming to balance security impact (severity) and remediation effort (complexity, size). This approach was validated through expert surveys and correlation analysis with historical remediation times [5].

## Results and Discussion

Our experimental evaluation yielded several key findings across classification accuracy and prioritization effectiveness [7].

### Classification Performance

The LSTM architecture demonstrated superior performance across all metrics [7]. The detailed results are presented in Table II.

Notably, the LSTM maintained strong performance across all vulnerability classes, with particular strength in detecting SQL injection (96% recall) and XSS (94% recall) vulnerabilities [4]. The BNN showed interesting properties for rare classes due to its uncertainty quantification [15], while the AE struggled with syntactic variations in code structure [17].

### Prioritization Effectiveness

The prioritization framework successfully identified high- risk vulnerabilities, with:

- 92% of critical-severity flaws in top 20% of rankings
- 85% precision for top-priority items
- 40% reduction in false positives compared to complexity- only ranking

Developer surveys indicated the prioritization scores reduced investigation time by approximately 35% compared to unranked vulnerability lists [6].

### Comparative Analysis

When evaluated against binary classification baselines, our multi-class approach showed only 1% absolute accuracy degradation while providing significantly more detailed output

[13]. This marginal difference demonstrates that fine-grained vulnerability analysis can be achieved without sacrificing detection performance.

## Limitations and Future Work

While this study demonstrates promising results, several limitations offer pathways for future research. First, our mod- els were trained on a curated dataset from SARD and select real-world applications; performance on massive, unstructured legacy enterprise PHP codebases may vary and warrants fur- ther investigation. Second, our taxonomy, while covering six critical vulnerability types, could be expanded to include other OWASP Top 10 risks like insecure deserialization or security misconfigurations. Third, the generalizability of the "safe code" class, which represents non-vulnerable patterns similar to our training set, to entirely novel coding paradigms requires further study. Finally, the computational cost of training these models, though not prohibitive (approximately 8 hours on a single NVIDIA V100 GPU for the LSTM), presents a barrier to entry for some organizations; future work will explore model distillation for lighter-weight deployment. Despite these limitations, this work provides a strong foundation for fine- grained, prioritized vulnerability detection in PHP.

## Conclusions

This research makes significant contributions to the field of automated vulnerability analysis through three key innovations [7]:

- Effective Multi-class Detection: We demonstrate that modern deep learning architectures, particularly LSTMs, can achieve high accuracy (94%) in fine-grained vulner- ability classification without sacrificing detection performance compared to binary approaches.
- Practical Prioritization: Our hybrid scoring system effectively guides remediation efforts by combining static code metrics with security severity assessments, reducing investigation time by 35%.
- PHP-specific Advancements: The developed models ad- dress a critical gap in vulnerability research while sup- porting one of the web’s most widely used languages.

The implications for software development practice are substantial. By providing developers with specific vulnerability classifications and intelligent prioritization, our system can significantly reduce the time and effort required for security remediation [6]. The PHP-specific focus ensures immediate applicability to a large portion of web applications [8].
Future work will focus on three directions:

- Expansion to additional programming languages
- Integration with continuous integration pipelines
- Real-time detection capabilities

The complete implementation and dataset are being pre- pared for open-source release to support further research in this critical area [8].

## Author Contributions

Melese Awoke: Conceptualization, methodology, implementation, writing—original draft.
Yitayal Tehone: Data curation, validation, analysis, writ- ing—review & editing.
Both authors read and approved the final manuscript.

## Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

## Data Availability

The datasets generated and/or analysed during the current study are available in the GitHub repository: https://github.com/MeleseAwoke/PHP-vulnerability-dataset. git

## References

1. OWASP, "OWASP Top 10: 2023," OWASP Foundation, 2023.
2. Y. Fang, Y. Liu, C. Huang, and L. Liu, "Fast and deep vulnerability detection using continuous learning," PLoS ONE, vol. 14, no. 12, p. e0226333, 2019.
3. H. Hanif, S. M. A. Iqbal, and M. N. M. Saad, "Software vulnerability identification using machine learning: A systematic literature review," J. Netw. Comput. Appl., vol. 191, p. 103146, 2021.
4. R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated vulnerability detection in source code using deep representation learning," in Proc. IEEE ICMLA, 2018, pp. 757–762.
5. R. Bhoure and A. Laddad, "A systematic approach to vulnerability management using prioritization and prediction," Int. Res. J. Eng. Technol., vol. 8, no. 5, pp. 1234–1240, 2021.
6. G. Tang, X. Meng, H. Wang, and Z. Chen, "Vulnerability detection using kernel extreme learning machine and feature learning," IEEE Access, vol. 9, pp. 64095–64105, 2021.
7. Y. Wu, D. Zou, W. Yang, W. J. Li, F. Cheng, and H. Jin, "Vulnera- bility detection with deep learning," in Proc. IEEE CCWC, 2021, pp. 0537–0542.
8. NIST, "Software Assurance Reference Dataset (SARD)," 2022. [Online].
9. Available: https://samate.nist.gov/SARD/
10. D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf, "Bugs as deviant behavior: A general approach to inferring errors in systems code," ACM SIGOPS Oper. Syst. Rev., vol. 35, no. 5, pp. 55–72, 2001.
11. O. Alhazmi, Y. Malaiya, and I. Ray, "Security vulnerability categories,"
12. in Proc. IASTED CNIS, 2006.
13. Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Comak, and L. Karacan, "Vulnerability prediction from source code using machine learning," IEEE Access, vol. 8, pp. 150672–150684, 2015.
14. G. Lin, J. Zhang, W. Luo, L. Pan, and Y. Xiang, "Software vulnerability discovery via

learning multi-domain knowledge bases," IEEE Trans. Dependable Secure Comput., vol. 18, no. 5, pp. 2469–2485, 2020.

15. Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vul- nerability identification by learning comprehensive program semantics via graph neural networks," in Adv. Neural Inf. Process. Syst., vol. 32, 2019.
16. S. Syed and A. A. Khan, "A hybrid approach for web vulnerability discovery and removal," in Proc. ICNTE, 2019, pp. 1–5. De Waal, J. M. E. T. E. R. Bosch, and P. P. J. L. C. M. Smits,
17. "Uncertainty quantification in deep learning for automated vulnerability detection," Inf. Fusion, vol. 64, pp. 85–96, 2020.
18. J. Pewny, F. Schuster, L. Bernhard, T. Holz, and C. Rossow, "Leveraging semantic signatures for bug search in binary programs," in Proc. 30th Annu. Comput. Secur. Appl. Conf., 2014, pp. 406–415.
19. P. Baldi, "Autoencoders, unsupervised learning, and deep architectures,"
20. J. Mach. Learn. Res., vol. 27, pp. 37–50, 2012.