**International Journal of Scientific Research and Management (IJSRM)**
**||Volume||13||Issue||07||Pages||2434-2445||2025||**
**Website: https://ijsrm.net ISSN (e): 2321-3418**
**DOI: 10.18535/ijsrm/v13i07.ec06**

# BayesPDGImVD: Bayesian Hyperparameter-Optimized Image-Based Vulnerability Detection via Program Dependency Graph Representation

**Xingquan Mao[1*], Zhangpei Huang[2]**

[1,2]School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China

**Abstract**
With the increasing complexity of software systems and widespread adoption of open-source components, traditional vulnerability detection approaches face significant bottlenecks in both efficiency and accuracy. Recent advances in machine learning have opened new avenues for intelligent vulnerability detection. This paper presents a Bayesian Hyperparameter-Optimized Image-Based Vulnerability Detection method via Program Dependency Graph Representation (BayesPDGImVD), which innovatively combines program dependency graph (PDG) image representation with Bayesian hyperparameter optimization to effectively overcome limitations of conventional detection methods. The implemented system performs static PDG extraction from C/C++ source code using the Joern analyzer, then constructs multi-channel image features by integrating Sent2Vec semantic embeddings with triple-node centrality metrics (degree, closeness, and Katz centrality). The CNN classifier employs Bayesian optimization to automatically tune critical parameters (learning rate, kernel size, dropout rate, etc.), completely eliminating manual parameter adjustment. Experimental results on the SARD benchmark dataset demonstrate outstanding performance: 86.43% detection accuracy and 80.38% F1-score, with 40% reduced performance fluctuation compared to non-optimized models, validating Bayesian optimization's effectiveness in enhancing model robustness and detection capability. Unlike existing approaches such as VulCNN, our key contribution lies in the organic integration of image-based representation with hyperparameter optimization mechanisms, providing a more interpretable and engineering-practical solution for source code vulnerability detection.

Keywords: Machine learning, Bayesian optimization, Program dependency graph, Image-based representation, Source code vulnerability detection.

## 1. Introduction

With the rapid advancement of information technology and the continuous expansion of software system scale, software vulnerabilities have become an increasingly critical challenge in the field of cybersecurity. Traditional vulnerability detection methods relying on manual auditing and rule-based approaches have demonstrated diminishing efficiency and accuracy when dealing with the complex, evolving, and large-scale modern software codebases. These methods are often constrained by expert experience and predefined rules, resulting in lengthy detection cycles, high costs, and frequent false positives/negatives - particularly when encountering novel or variant vulnerabilities, where their limitations become markedly apparent [1][2].

In recent years, significant progress in deep learning technologies has brought new opportunities for source code vulnerability detection. Deep neural networks can autonomously learn syntactic structures, control flow characteristics, and underlying semantic patterns from code, effectively addressing the feature engineering limitations of traditional approaches. However, standalone deep learning models still face challenges when processing complex code semantics and structures, including insufficient generalization capability, sensitivity to noise, and difficulties in comprehensively capturing multi-dimensional features. Particularly, while image-based source code vulnerability detection methods show promising potential, key research challenges remain regarding how to more effectively preserve and represent program dependency relationships during the image transformation process, and how to optimize model performance to adapt to diverse code characteristics [3][4].

To address these challenges, this paper proposes a Bayesian Hyperparameter-Optimized Image-Based Vulnerability Detection via Program Dependency Graph Representation (BayesPDGImVD). The proposed method aims to transform source code Program Dependency Graphs (PDGs) into image representations and perform vulnerability detection using Convolutional Neural Networks (CNNs), while incorporating Bayesian optimization to automatically tune critical hyperparameters. This approach seeks to enhance model robustness and generalization capability while maintaining high detection accuracy.

The main contributions of this paper can be summarized as follows:

(1) We introduce a Bayesian optimization algorithm to automatically tune critical hyperparameters in the CNN model (including learning rate, kernel size, dropout rate, etc.), which significantly enhances the model's generalization capability and robustness across different code samples and datasets. This approach reduces reliance on empirical parameter tuning and effectively mitigates both false negative and false positive issues.

(2) We develop a Bayesian hyperparameter-optimized image-based vulnerability detection framework via program dependency graph representation. Building upon existing image-based vulnerability detection frameworks, we incorporate Bayesian optimization and conduct systematic experiments on both real-world and synthetic datasets. The results demonstrate the effectiveness of the Bayesian optimization strategy in improving model stability and detection accuracy, providing a practical improvement path for future research.

The organization of the subsequent sections is as follows: Section 2 reviews related literature; Section 3 details the design of the BayesPDGImVD framework; Sections 4 presents the experimental design and result analysis; Section 5 discusses validity threats; Section 6 concludes the paper.

## 2. **Literature Review**

With the continuous increase in software system complexity, security vulnerabilities in source code have become an increasingly serious threat. Traditional vulnerability detection methods, such as manual code auditing and signature-based detection, suffer from low efficiency and difficulty in adapting to emerging vulnerability types. In recent years, machine learning and deep learning technologies have demonstrated significant potential in the field of vulnerability detection, particularly showing remarkable progress in program representation and graph neural network applications.

### 2.1 Source Code Vulnerability Detection Techniques

Source code vulnerability detection serves as a critical component in software security assurance. Early detection methods primarily relied on static analysis tools that identified known vulnerability patterns through pattern matching or data flow analysis. For instance, Srinivasa et al. proposed an abstract interpretation-based static analysis method for detecting buffer overflows in C programs [5]. However, these approaches often face challenges of high false positive/negative rates and struggle to identify complex logical vulnerabilities.

With the advancement of machine learning techniques, researchers began combining source code features with classification algorithms. For example, Peng et al. employed program slicing to extract features and utilized support vector machines for vulnerability classification [6]. The rise of deep learning brought new breakthroughs in vulnerability detection, where convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have demonstrated exceptional performance in learning deep semantic features of code. Research teams led by Bilgin and Wu have proposed CNN-based vulnerability detection methods that capture spatial features by transforming source code into image representations [7][8]. Nevertheless, these methods still exhibit limitations in processing program structural information.

## 2.2 Application of Program Dependency Graphs in Vulnerability Detection

Program Dependency Graphs (PDGs), as graph structures capable of effectively representing program control flow and data flow, have been widely adopted in vulnerability detection. PDGs can reveal dependency relationships between statements within programs, providing rich contextual information for analyzing program behavior and identifying potential vulnerabilities. Yamaguchi et al. utilized PDGs to construct program representations and detected specific types of vulnerabilities through graph traversal algorithms [9].

In recent years, the emergence of Graph Neural Networks (GNNs) has made it possible to directly learn from PDGs. For instance, Li et al. proposed a vulnerability detection method based on Graph Convolutional Networks (GCNs), which learns node representations by propagating information across PDGs [10]. Zhou et al. further explored the potential of Graph Attention Networks (GATs) in learning node representations from PDGs for vulnerability detection [11].

Although GNNs excel at capturing graph structural information, challenges remain in effectively integrating different types of dependency information and efficiently processing large-scale PDGs.。

## 2.3 Image-Based Vulnerability Detection Techniques

The conversion of source code or its intermediate representations (e.g., Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), Program Dependency Graphs (PDGs)) into images, followed by vulnerability detection using deep learning models from computer vision (particularly Convolutional Neural Networks), has emerged as a promising research direction in recent years. The core advantage of this approach lies in leveraging CNN's powerful spatial feature extraction capability to capture local patterns in code.

Early explorations primarily focused on directly treating source code text as one-dimensional sequences or two-dimensional grids (e.g., pixelated code text) for modeling [12][13][14]. Subsequent work, however, demonstrated that transforming structured code representations (e.g., PDGs) into multi-dimensional feature images can more effectively integrate semantic and structural information, thereby improving model recognition performance. A significant breakthrough came from Li et al.'s VulCNN method[8]. VulCNN represents PDG nodes (e.g., statements) as semantic vectors through word embeddings and combines node centrality metrics (e.g., degree centrality, closeness centrality, Katz centrality) to construct multi-channel feature maps, thereby converting PDGs into 3D tensors suitable for CNN input. This image-based representation effectively fuses code's semantic and structural information, demonstrating strong performance in vulnerability detection tasks.

However, a key limitation of VulCNN and similar methods lies in their CNN models' heavy dependence on hyperparameter settings (e.g., learning rate, kernel size, dropout rate). Typically, tuning these hyperparameters relies on researchers' experience and extensive trial-and-error, which is not only inefficient but also fails to guarantee optimal model performance or stable generalization capability.

## 2.4 The Role of Hyperparameter Optimization in Deep Learning

Traditional hyperparameter optimization methods, such as Grid Search and Random Search, suffer from high computational costs and low efficiency, particularly in high-dimensional hyperparameter spaces. Bischl et al. provided a detailed explanation of the principles and limitations of Grid Search and Random Search [15]。

To improve the efficiency of hyperparameter optimization, Bayesian optimization has been introduced as a more intelligent Sequential Model-Based Optimization (SMBO) method. Bayesian optimization constructs a probabilistic surrogate model (typically a Gaussian Process) of the objective function (usually a performance metric on the validation set, such as accuracy or F1-score) and uses an acquisition function (e.g., Expected Improvement (EI) or Upper Confidence Bound (UCB)) to guide the selection of the next hyperparameter combination for evaluation. This approach enables the identification of near-optimal hyperparameters within a limited number of evaluations. Snoek et al. systematically described the fundamental principles of Bayesian optimization and its successful applications in hyperparameter tuning for machine learning models [16].

Although Bayesian optimization has achieved success in various domains, including some traditional vulnerability detection models based on code sequences or simple code attribute features [17], its application to complex deep learning models that rely on graph-structured image representations (e.g., the multi-channel PDG images used in VulCNN) and its end-to-end optimization for vulnerability detection tasks remain insufficiently explored.

### 2.5 Positioning of This Work

Existing research has made significant progress in leveraging deep learning for vulnerability detection, yet several challenges remain: (1) how to optimize hyperparameters of deep learning models to improve detection performance and generalization capability, and (2) how to construct a hyperparameter-optimized, image-based vulnerability detection method using program dependency graphs (PDGs) to validate the effectiveness of hyperparameter optimization strategies in enhancing model stability and detection accuracy.

This work aims to bridge these gaps by combining the structural advantages of PDGs with the efficiency of Bayesian hyperparameter optimization. We propose a novel vulnerability detection method that not only improves detection accuracy but also reduces the complexity of model tuning.

## 3. The proposed approach

This study proposes a Bayesian Hyperparameter-Optimized Image-Based Vulnerability Detection method via Program Dependency Graph Representation, named BayesPDGImVD. The core concept of this method lies in introducing a Bayesian optimization algorithm into the existing image-based vulnerability detection framework VulCNN[8] to automatically adjust critical hyperparameters in the convolutional neural network model (including learning rate, kernel size, dropout rate, etc.). This approach aims to significantly enhance the model's generalization capability and robustness across different code samples and datasets, reduce reliance on empirical parameter tuning, and effectively mitigate false negative and false positive issues.

Specifically, this research precisely characterizes the deep semantic and structural information of source code through Program Dependence Graphs (PDGs) and transforms PDGs into multi-channel image representations. This transformation is designed to fully leverage the outstanding pattern recognition capabilities of convolutional neural networks in visual domains, thereby achieving efficient vulnerability discrimination. To address the empirical dependency of critical hyperparameter configuration in deep learning model training and its potential impact on model performance stability, this framework deeply integrates a Bayesian optimization algorithm. This algorithm enables automated search and optimization of model hyperparameters, significantly improving the robustness and generalization capability of the detection model.

The entire methodological workflow is systematically divided into three interconnected and progressively advanced main stages: (1) source code semantic parsing and structural representation, (2) graph-to-multi-channel-image mapping, and (3) Bayesian-optimized CNN classification model construction and training.

In the source code semantic parsing and structural representation phase, this study first performs rigorous preprocessing and normalization operations on the input C/C++ source code, which serves as the critical foundation for ensuring consistency in subsequent analysis and the model's generalization capability.

The normalization process encompasses three levels: first, the complete removal of all comments from the source code, as they do not carry program execution semantics; second, the systematic one-to-one mapping of user-defined variable names to unified symbolic identifiers (e.g., VAR1, VAR2, etc.); similarly, user-defined function names are also mapped one-to-one to unified symbolic identifiers (e.g., FUN1, FUN2, etc.). This symbol unification strategy not only effectively eliminates noise introduced by naming convention differences but also strictly preserves the program's original logical structure and data/control flow semantics.

Following normalization, this study employs the mature open-source code analysis tool Joern [9] to perform in-depth static parsing of the preprocessed source code. Joern can accurately extract function-level Program Dependence Graphs (PDGs), where nodes precisely correspond to each line of valid code statements in the function, and edges clearly depict two critical types of dependency relationships between nodes: data dependencies and control dependencies. Data dependency edges explicitly illustrate the definition and usage relationships of variables in the program—how the values of variables defined or modified by one statement are read or relied upon by subsequent statements. Control dependency edges precisely express the control logic of program execution flow, indicating whether (or how often) the execution of a statement is conditioned by the result of specific branching statements (e.g., if, for, while, etc.). By constructing the PDG, the implicit complex semantic logic and structural information in the source code are explicitly and structurally captured and presented, laying a solid foundation for subsequent feature extraction.

In the graph-to-multi-channel-image mapping phase, this study efficiently and informatively transforms the non-Euclidean graph-structured data (PDG) into Euclidean image data suitable for CNN processing. This conversion process consists of two core steps: node semantic vectorization and graph structural feature quantification with multi-channel fusion. First, for each node in the PDG (i.e., a line of code), we treat it as an independent semantic unit (analogous to a sentence in natural language). To capture its semantic information, we employ the pre-trained Sent2Vec model [18] to embed each line of code. Sent2Vec is an unsupervised learning-based sentence vector representation model that can convert a line of code text into a fixed-dimensional (set to 128 dimensions in this study) dense real-valued vector. This vector effectively encodes the semantic content of the line, such as the type of operation performed, the variables or constants involved, and other critical information.

To effectively integrate the structural importance reflected by the connectivity relationships between nodes in the PDG, this study draws on existing work (Wu et al. referenced centrality metrics from social network analysis to measure the role of each node in the program dependence graph from multiple perspectives [8]). Specifically, three complementary centrality metrics are adopted to characterize the structural role of nodes: degree centrality measures the number of direct connections, reflecting the node's local connectivity strength; closeness centrality measures the average shortest path distance from a node to all other nodes, indicating its centrality in the global topology; and Katz centrality further incorporates the influence of indirect connections by weighting path lengths to assess a node's propagation capability and influence range. These three metrics collectively provide a multi-dimensional evaluation of the structural-semantic importance of each line of code in the program.

Subsequently, inspired by the RGB three-channel structure in image processing, the three centrality metrics are treated as three feature channels and fused with the node's semantic vector. Specifically, for each node (i.e., a code statement), it is first embedded into a fixed-dimensional (128-dimensional) dense vector using the pre-trained Sent2Vec model to represent its semantic information. This semantic vector is then element-wise multiplied with the node's three centrality values, thereby combining the semantic features with the node's structural role information in the program.

After this processing, the entire PDG is transformed into a three-dimensional tensor with the shape (3, N, 128), where 3 represents the three centrality channels, N is the number of code statements (i.e., PDG nodes) in the function, and 128 is the dimensionality of the semantic embedding vector. This tensor can be regarded as an image-like representation of the source code in structural and semantic dimensions and serves as input to the CNN. Leveraging its powerful local pattern learning capability, the CNN can capture potential semantic dependencies and structural patterns in the program. It is worth emphasizing that this strategy of converting program dependence graphs into multi-channel images is not novel to this study but is instead integrated and optimized based on the existing VulCNN framework to support subsequent Bayesian hyperparameter search and model training.

In the Bayesian-optimized CNN classification model construction and training phase, this method focuses on training a high-performance binary classifier (vulnerable/non-vulnerable) while addressing the challenging issue of hyperparameter optimization in deep model training. An improved TextCNN architecture [19] is adopted as the core classification model. The model takes the aforementioned three-channel image (3, N, 128) as input. The core of the model consists of multiple groups of convolutional layers with varying scales, where each group employs convolutional kernels of specific sizes (e.g., 1*128, 2*128, 3*128) to perform sliding convolution operations along the vertical dimension of the image (i.e., the node sequence direction in the PDG, dimension N). Different kernel sizes effectively capture local patterns and dependency relationships at varying granularities in the code line sequence (e.g., single-line operations, interactions between two adjacent lines, or contextual relationships spanning three lines). After convolution, the ReLU activation function [20] is applied to introduce nonlinear transformation capability, followed by pooling layers for feature dimensionality reduction and key feature extraction. The feature maps extracted by all convolutional layers are concatenated into a long feature vector, fed into multiple fully connected layers, and finally passed through a Softmax layer to output the probability distribution of whether the sample contains a vulnerability.

During model training, this study employs cross-entropy [21] as the loss function and the AdamW optimizer [22] for parameter updates to ensure stable and efficient training. To effectively improve model performance while reducing the subjectivity and high cost of manual hyperparameter tuning, a Bayesian optimization algorithm is introduced as the core hyperparameter search strategy. Specifically, the Optuna framework [23] is used to construct an automated hyperparameter optimization workflow for determining the optimal combination of multiple critical parameters in the CNN.

The optimization scope includes the combination of kernel sizes (e.g., [1, 2] or [1, 2, 3]), the number of kernels (affecting the expressive power of feature extraction), dropout rate (for controlling overfitting), training batch size, and learning rate (typically set on a logarithmic scale, e.g., from 1e-4 to 1e-3). Bayesian optimization constructs a probabilistic surrogate model of the objective function and iteratively updates it based on historical evaluation results, effectively avoiding the computational overhead of extensive sampling in grid search and random search. Each iteration selects the hyperparameter combination predicted to have the highest potential for performance improvement, significantly enhancing search efficiency and convergence speed. In this study, 30 optimization iterations are performed to find a near-global optimum within reasonable resource consumption.

After determining the optimal hyperparameter configuration, the CNN model is fully trained on the complete training set, and its stability and generalization capability are evaluated using five-fold cross-validation [24]. The training process runs for 200 epochs until model convergence. The final CNN model, combined with the automatically tuned optimal hyperparameters via Bayesian optimization, forms the core engine of the proposed source code vulnerability detection method, demonstrating strong detection performance and robust model generalization.

## 4. Experiments

This experiment focuses on the optimization and performance evaluation of the vulnerability detection model, aiming to answer the following research question (RQ) through systematic experimentation:

**·RQ: Can the Bayesian optimization algorithm effectively improve the accuracy and generalization performance of vulnerability detection when automatically tuning hyperparameters of convolutional neural network models?**

## 4.1. **Experimental settings**

The dataset used in this study is derived from the publicly available vulnerability dataset provided by VulCNN [8]. This dataset is constructed based on the Software Assurance Reference Dataset (SARD), comprising 12,303 vulnerable samples and 21,057 non-vulnerable samples. During experimentation, the dataset was partitioned in an 8:2 ratio, with 80% allocated as the training set and the remaining 20% reserved as the test set, ensuring objective model evaluation and generalizable, reliable experimental results.

The effectiveness of BayesPDGImVD can be evaluated using performance metrics including accuracy, recall, and F1-score. The specific formulas for each metric are presented in Table 1, where TP denotes true positives, TN represents true negatives, FP indicates false positives, and FN signifies false negatives.

Table 1 Experimental Metrics

| Metrics | Formulas |
|---|---|
| Accuracy | $A=\dfrac{TP+TN}{TP+FP+TN+FN}$ |
| Recall | $FNR=\dfrac{TP}{TP+FN}$ |
| F1 score | $F1=\dfrac{2*P(1-FNR)}{P+(1-FNR)}$ |

## 4.2. Experiments to answer RQ

The core focus of this research lies in investigating the fundamental performance enhancement effec t of Bayesian optimization algorithms on convolutional neural network-based vulnerability detection models. To this end, we designed and implemented a rigorous experimental procedure aimed at tran scending superficial metric comparisons to deeply analyze the intrinsic impact of the optimization m echanism. The experiments systematically compared two models - one with Bayesian optimization ( BayesPDGImVD) and another with manually configured hyperparameters (Baseline) - in terms of th eir training dynamics and final performance on identical datasets. The model training spanned 200 e pochs, with detailed recording of key metrics including training/validation loss, accuracy, and F1-sco re for each epoch, which were transformed into structured data and visualized through custom script s.

The training and validation loss curves are shown in Figure 1 (optimized model) and Figure 2 (non-optimized model). The Bayesian-optimized model demonstrated significantly superior learning capability and convergence characteristics. Its training loss rapidly decreased from an initial 0.50 to around 0.25, while validation loss stabilized at approximately 0.28 (Figure 1). This not only indicat es effective parameter updates but, more crucially, reveals that the model achieved balanced fitting on both training and validation sets, successfully avoiding risks of overfitting or underfitting. In con trast, the non-optimized baseline model (Figure 2) exhibited prolonged training and validation losses above 0.3, with slow convergence and evident plateauing, suggesting the model was constrained by suboptimal hyperparameter configurations and unable to fully exploit data potential for good genera lization.

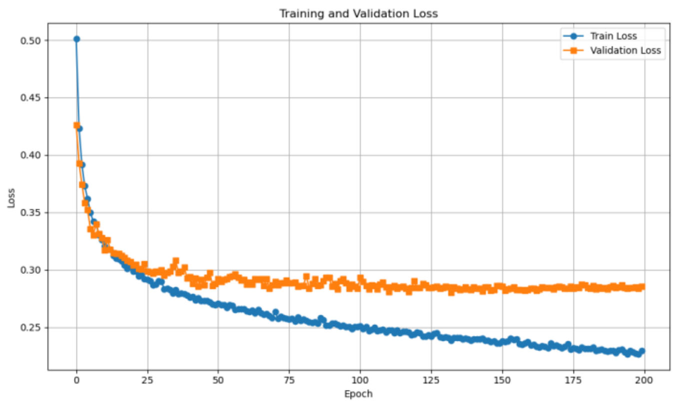

Figure 1 Training and validation loss curves of the hyperparameter-optimized model

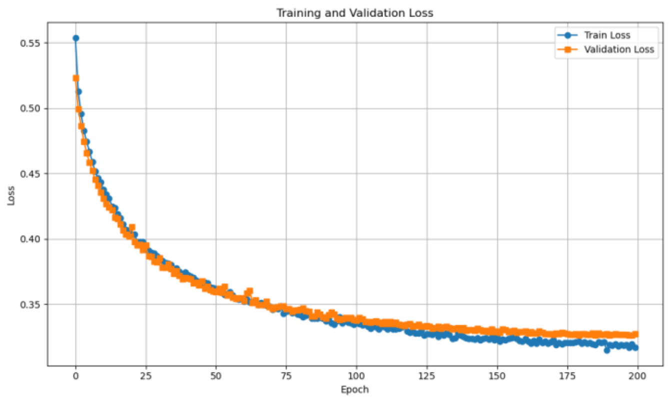

Figure 2 Training and validation loss curves of the non-optimized model

Performance metric comparisons further confirmed the profound value of Bayesian optimization. The optimized model's training accuracy steadily increased from 72.7% to 86%, while validation a ccuracy improved from 78.4% to 82% (Figure 3). Notably, the minimal gap (≈3%) between training and validation accuracy strongly demonstrates the model's exceptional generalization capability, indi cating that the learned feature patterns possess high universality rather than being mere mechanical memorization of training data. The F1-score, as a balanced metric incorporating both precision and recall, provides even more compelling evidence. The optimized model's macro-average F1-score on t he validation set remained stable within the 74%-78% range (Figure 4), ultimately reaching 80.38%, significantly outperforming the non-optimized model's F1-score which fluctuated violently around 7 0% (Figure 5).

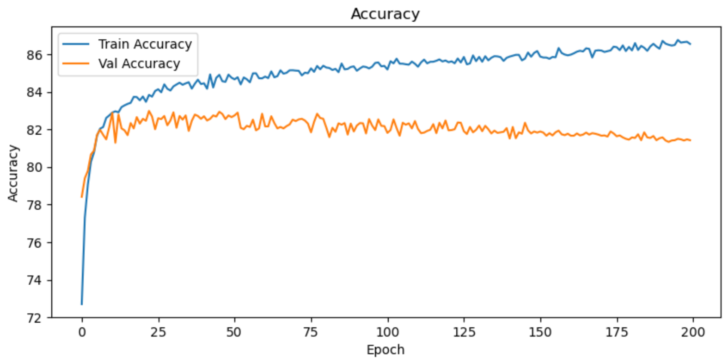

Figure 3 Accuracy curves of the hyperparameter-optimized model

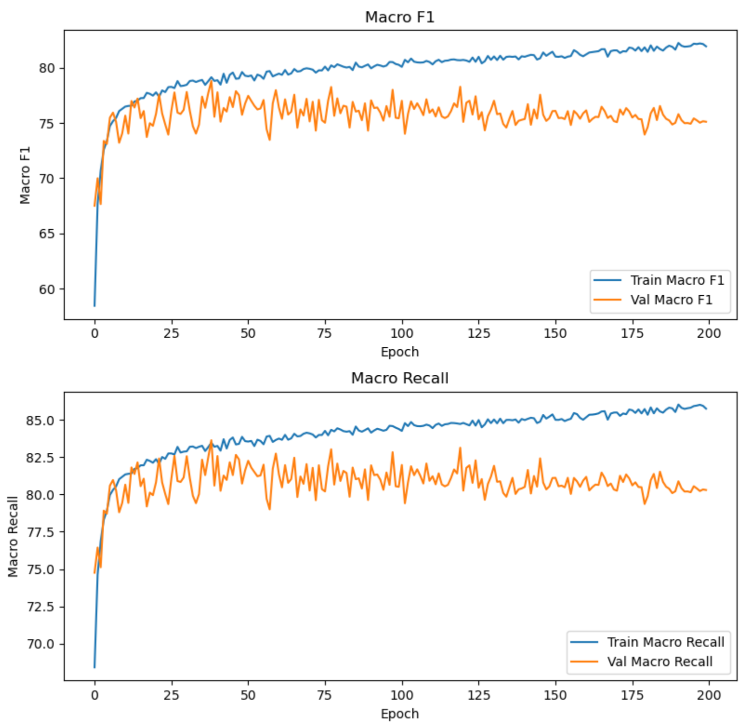

Figure 4 F1-score and recall curves of the hyperparameter-optimized model

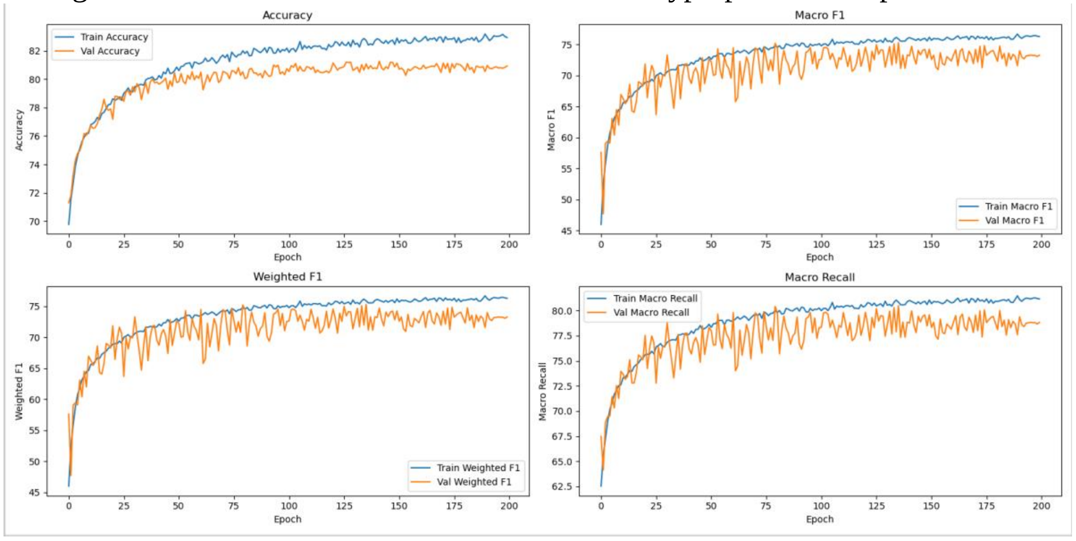

Figure 5 Performance charts of the non-optimized model

The ≈40% reduction in fluctuation amplitude profoundly reflects the fundamental enhancement o f model robustness through Bayesian optimization - the model's performance is no longer sensitive t o specific training batches or data subsets, demonstrating consistent discriminative capability across different samples. Meanwhile, the significant improvement in recall rate (training: 68%→84%; valida tion: 74.7%→81%) indicates substantial strengthening of the model's ability to capture positive cases (vulnerable samples), helping reduce false negatives in practical applications.

Delving into the root causes of performance differences, the key lies in Bayesian optimization's adaptive search process for core model hyperparameters. Compared to the baseline model's empiric al settings of fixed (1,2,3) kernel sizes, 32 filters, 0.5 dropout rate, batch size 32, and learning rate 0.001, the Bayesian optimization algorithm (implemented via Optuna) efficiently explored complex interaction effects of hyperparameter combinations within predefined search spaces (e.g., kernel size combinations, number of filters, dropout rate range, learning rate log space, etc.). By constructing s urrogate models and acquisition functions, it automatically located optimal configurations better suite d to the current task characteristics (source code PDG image representation) and data distribution (S ARD vulnerability dataset). For instance, the algorithm might have discovered more effective kernel size combinations to capture code dependency patterns at different granularities, adjusted dropout r ates to achieve better balance between model capacity and regularization strength, or optimized lear ning rates to accelerate convergence while avoiding local optima. This automated, data-driven optimi zation process fundamentally addresses the blindness, high cost, and difficulty in reproducing optima l solutions inherent in manual tuning, serving as the core driver for the model's achieved high stabi lity, strong generalization capability, and significantly improved overall performance (accuracy 86.43 %, F1 80.38%).

In conclusion, the experimental results provide clear and compelling answers to the research qu estions, verifying that using Bayesian optimization to automatically tune hyperparameters of CNN m odels not only significantly improves vulnerability detection accuracy (evidenced by higher accuracy and F1-scores) but, more importantly, fundamentally enhances model robustness and generalization capability (manifested as substantially reduced validation metric fluctuations and small training/valida tion performance gaps). This effectively mitigates the model instability, poor generalization, and pot ential false negative/positive issues caused by traditional experience-based parameter tuning. The find ings also validate the effectiveness and necessity of deeply integrating Bayesian optimization into th e program dependency graph image-based vulnerability detection pipeline.

## 5. Threats to validity

Although the proposed BayesPDGImVD method demonstrates superior performance in experiments, its potential validity threats require careful consideration. The primary threat stems from the representativeness and generalization capability of the dataset. The experiments used the SARD benchmark dataset, which is widely adopted in vulnerability detection research. However, its samples primarily contain synthetic vulnerabilities, differing from the complex vulnerability patterns found in real-world industrial projects. Although we enhanced the model's generalization through normalization techniques, including the generalization of variable and function names, its adaptability to highly heterogeneous coding styles, third-party library calls, and novel vulnerabilities in real-world scenarios requires further validation. Future work should include testing on real-world project data to assess the model's practical utility.

Secondly, dependencies in the technical stack may introduce bias. The generation of Program Dependency Graphs (PDGs) relies entirely on the accuracy of the Joern static analysis tool. If this tool has limitations in parsing certain complex syntactic structures—such as pointer arithmetic, macro expansion, or concurrent code—it may lead to distorted graph structures, thereby affecting the quality of subsequent image-based representations. Similarly, the semantic embedding of nodes employs a pre-trained Sent2Vec model, whose representation capability is constrained by the alignment between its training corpus and code text. Although Sent2Vec performs robustly in natural language processing tasks, its ability to capture code-specific syntactic features, such as type constraints and scope nesting, still requires verification through more detailed ablation experiments.

Additionally, the constraints in Bayesian optimization may impact the effectiveness of hyperparameter search. The experiment set 30 iterations to balance search efficiency and performance, but in high-dimensional hyperparameter spaces, the search for optimal configurations—such as kernel combinations, dropout rates, and learning rates—may not have fully converged. While the surrogate model, using Gaussian processes, significantly improved search efficiency, its probability-based assumptions may introduce approximation errors in highly non-convex loss function spaces. Moreover, the optimization objective focused solely on the validation set's F1-score without explicitly constraining computational resource consumption or model complexity, potentially leading to excessive resource overhead in practical engineering applications.

Finally, the method's generalization capability has certain boundaries. The experiments only validated the approach on C and C++ code, whereas modern software often involves multi-language hybrid development, such as Python binding C modules. The current technical stack, including Joern and Sent2Vec, has limited support for non-C language families. Furthermore, whether the image-based strategy is applicable to dynamically typed languages like JavaScript or scripting languages like Python remains unexplored. The model primarily focuses on detecting memory-related vulnerabilities, particularly buffer overflow vulnerabilities, which are prevalent in the SARD dataset. However, its sensitivity to logical vulnerabilities, such as privilege escalation or race conditions, has not been thoroughly evaluated. Future work should expand the coverage of vulnerability types to strengthen the conclusions' generalizability.

## 6. Conclusions

This paper proposes BayesPDGImVD—a program dependency graph image-based vulnerability detection method integrated with Bayesian hyperparameter optimization, effectively addressing the dual challenges of insufficient generalization in traditional vulnerability detection techniques and performance instability in deep learning models. By parsing C/C++ source code into structured PDG representations and constructing multi-channel image features through Sent2Vec semantic embeddings and node centrality metrics, the method fully leverages the advantages of convolutional neural networks (CNNs) in local pattern recognition. More importantly, it introduces, for the first time, a Bayesian optimization algorithm to automate the search for critical hyperparameters, significantly reducing reliance on manual parameter tuning. Experimental results on the SARD benchmark dataset demonstrate that BayesPDGImVD achieves a final accuracy of 86.43% and an F1-score of 80.38%, with a ≈40% reduction in F1-score fluctuation on the validation set compared to the non-optimized model. These findings robustly validate its effectiveness in improving detection precision, enhancing model robustness, and mitigating false negatives/positives.

The core contribution of this method lies in establishing a closed-loop framework of "program structure → image representation → adaptive optimization." On one hand, the multi-channel image-based strategy for PDGs (semantic + structural centrality) provides a novel approach to the visualization and interpretability of code features. On the other hand, the deep integration of Bayesian optimization with CNNs offers an efficient and reproducible engineering solution to the hyperparameter configuration challenges of complex deep learning models. Experimental results confirm that this framework not only surpasses the performance ceiling of baseline models but also provides technical support for the deployment of industrial-grade vulnerability scanning tools through its stable generalization capability.

## References

1. Uddin MN, Zhang Y, Hei XS. Deep Learning Aided Software Vulnerability Detection: A Survey. ArXiv [Internet]. 2025;abs/2503.04002.
2. Harzevili NS, Belle AB, Wang J, et al. A Survey on Automated Software Vulnerability Detection Using Machine Learning and Deep Learning. ArXiv [Internet]. 2023;abs/2306.11673.
3. Peng T, Gui L, Sun Y. VulMCI : Code Splicing-based Pixel-row Oversampling for More Continuous Vulnerability Image Generation. ArXiv [Internet]. 2024;abs/2402.18189.
4. Fedorchenko E, Novikova E, Fedorchenko A, et al. An Analytical Review of the Source Code Models for Exploit Analysis. Inf. 2023;14:497.
5. Srinivasa G. Abstraction-Based Static Analysis of Buffer Overruns in C Programs. 2003. Available from: https://api.semanticscholar.org/CorpusID:14270976.
6. Peng B, Su P, Liu Z, et al. VulSimple: A Vulnerability Detection Framework Based on Simple Backward Slicing. Proc 4th Int Conf Artif Intell Comput Eng [Internet]. 2023;
7. Bilgin Z. Code2Image: Intelligent Code Analysis by Computer Vision Techniques and Application to Vulnerability Prediction. ArXiv [Internet]. 2021;abs/2105.03131.
8. Wu Y, Zou D, Dou S, et al. VulCNN: an image-inspired scalable vulnerability detection system. Proc 44th Int Conf Softw Eng [Internet]. New York, NY, USA: Association for Computing Machinery; 2022. p. 2365–2376. Available from: https://doi.org/10.1145/3510003.3510229.
9. Yamaguchi F, Golde N, Arp D, et al. Modeling and Discovering Vulnerabilities with Code Property Graphs. 2014 IEEE Symp Secur Priv. 2014. p. 590–604.

10. Li Y, Wang S, Nguyen TN. Vulnerability detection with fine-grained interpretations. Proc 29th ACM Jt Meet Eur Softw Eng Conf Symp Found Softw Eng [Internet]. 2021;
11. Zhou L, Huang M, Li Y, et al. GraphEye: A Novel Solution for Detecting Vulnerable Functions Based on Graph Attention Network. 2021 IEEE Sixth Int Conf Data Sci Cyberspace DSC. 2021;381–388.
12. Li Z, Zou D, Xu S, et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. CoRR [Internet]. 2018;abs/1801.01681.
13. Li Z, Zou D, Xu S, et al. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. IEEE Trans Dependable Secure Comput. 2018;19:2244–2258.
14. Zhou Y, Liu S, Siow J, et al. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks. Proc 33rd Int Conf Neural Inf Process Syst. Red Hook, NY, USA: Curran Associates Inc.; 2019.
15. Bischl B, Binder M, Lang M, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. Wiley Interdiscip Rev Data Min Knowl Discov [Internet]. 2021;13.
16. Snoek J, Larochelle H, Adams RP. Practical Bayesian Optimization of Machine Learning Algorithms. Neural Inf Process Syst [Internet]. 2012. Available from: https://api.semanticscholar.org/CorpusID:632197.
17. Turner R, Eriksson D, McCourt MJ, et al. Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. Neural Inf Process Syst [Internet]. 2021. Available from: https://api.semanticscholar.org/CorpusID:233324399.
18. Pagliardini M, Gupta P, Jaggi M. Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features. North Am Chapter Assoc Comput Linguist [Internet]. 2017. Available from: https://api.semanticscholar.org/CorpusID:16251657.
19. Kim Y. Convolutional Neural Networks for Sentence Classification. Conf Empir Methods Nat Lang Process [Internet]. 2014. Available from: https://api.semanticscholar.org/CorpusID:9672033.
20. Nair V, Hinton GE. Rectified Linear Units Improve Restricted Boltzmann Machines. Int Conf Mach Learn [Internet]. 2010. Available from: https://api.semanticscholar.org/CorpusID:15539264.
21. Shannon CE. The mathematical theory of communication. 1950. Available from: https://api.semanticscholar.org/CorpusID:125327631.
22. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization [Internet]. 2017. Available from: https://arxiv.org/abs/1412.6980.
23. Akiba T, Sano S, Yanase T, et al. Optuna: A Next-generation Hyperparameter Optimization Framework. Proc 25th ACM SIGKDD Int Conf Knowl Discov Data Min [Internet]. 2019;
24. Neal RM. Pattern Recognition and Machine Learning. J Electron Imaging [Internet]. 2006. Available from: https://api.semanticscholar.org/CorpusID:31993898.