Natural Resources for Human Health
eISSN: 2583-1194

Original Research

Received 29 May 2026
Revised 21 June 2026
Accepted 18 July 2026

Natural Resources for Human Health 2026, Vol. 6 (10s): 539–550

KEYWORDS:
Automated, Cyber Threat Detection, Proactive Defense, Prediction, and Intelligence Classification

An Intelligent BiGRU-GAP-Dense Framework for Automated Cyber Threat Detection and Proactive Defense

Prasad A. Lahare[1]*, Manoj A. Wakchaure[2]

[1]Research Scholar, Department of Computer Engineering , MET BKC IOE, Savitribai Phule Pune University, Pune, India. prasadlahare7@gmail.com
[2]Research Supervisor, Department of Computer Engineering , AVCOE, Savitribai Phule Pune University, Pune, India. manoj13apr@gmail.com

ABSTRACT: The continuous and rapid growth as well as expansion of software systems has significantly increased the chances of hidden vulnerabilities, which is a serious issue in cyber threat detection. The traditional static analysis tools usually fail to detect deeply embedded context-based vulnerabilities. To tackle these challenges, we introduce an integrated deep learning based automated framework cyber threat identification of source code which integrates Bidirectional Gated Recurrent Units (BiGRU), GAP and Dense layers with ReLU function. The main aim of proposed system is to automatically analyze source code snippets and determine whether it is vulnerable or non-vulnerable and if vulnerability is found then it also detects which type of vulnerability either XSS or Command Injection. For that we use Vulnerability Fix Dataset, a publicly available Kaggle dataset containing real-world code in multiple programming languages. To further enrich the training dataset, we extracted additional Python source code from open-source repositories such as Keras and The Algorithms using the PyDriller framework. Preprocessing comprises tokenization, the usage of statistical features, and cleaning of the code to improve quality of input model. The code sequences are then forwarded to created BiGRU- GAP to capture semantic and syntactic patterns. Experiments show that better classification accuracy and better generalization performance can be achieved by the developed network trained with a well-tuned pipeline and on high-performance computing platform. This paper proposed an efficient end-to-end holistic approach for proactive software vulnerability detection by using deep learning.

I. INTRODUCTION

The rapid expansion of open-source libraries and software development and complex interconnected systems is causing production code to be more susceptible to inadvertently introduced software vulnerabilities. Such bugs are widely abused by attackers like Cross Site Scripting (XSS) and Command Injection that may lead into disastrous security risks. Conventional static analysis tools and rule-based vulnerability scanners struggle to detect context- sensitive or syntactically variant security vulnerabilities, especially in large code bases [1]. With the latest developments in deep learning, it is becoming more feasible to automatically detect software vulnerabilities. For instance, RNN based model, like LSTM or GRUs, have been popular and successful in learning the sequential and syntactical programming languages [2][3]. Existing approaches, however, fail to model bidirectional dependencies and the majority of them cannot efficiently learn from variable- length code sequences without overfitting. The present paper we present a hybrid Deep neural network architecture, where we combine Bi-GRU, Global Average Pooling and Dense-ReLU layers to combat these challenges. This architecture is crafted to learn forward and backward context-dependences between code tokens and extract global sequence features while enabling the robustness of classification. We further extend our model with a preprocessing pipeline to mine and preprocess real-world Python code

from public repositories by means of the PyDriller tool.

The listed are the key contributions of the work:

- A novel BiGRU-based approach for automatic detection of source code vulnerabilities across multiple programming languages
- A comprehensive preprocessing pipeline using real-world multiple language code and structured tokenization.
- Performance evaluation on a labeled vulnerability dataset with evidence of improved accuracy and generalization.

The remainder of the paper structure is - Section second describes related work, Section third presents the hybrid system methodology and architectural design, Section four describes evaluation setup and results, and Section five will be conclusion and future work.

II. RELATED WORK

Wang et al.[4] suggest a 1D convolutional neural network for end-to-end traffic classification purpose. Which automatically learned different features directly from raw encrypted traffic data without any manual feature extraction. Results show that classification accuracy. P. Zeng et al. [5] provides a detailed review of deep learning techniques for software vulnerability classification. The paper classifies existing works by input representations (e.g., token, AST, bytecode), learning models (CNN RNN, GNN), and application domains. The authors discuss the inherent limitations of traditional static and dynamic analysis approaches tools along with it system propose that hybrid and graph-based models may be used for further development. Arıkan et al. [6] introduce a deep learning-enabled approach to automated threat intelligence generation for closed-source software vulnerabilities. Their approach extracts feature from compiled code and leverages a neural classifier to detect and report suspicious forks. The work shows a high detection rate on real malware data sets and tackles malware detection on proprietary software not accompanied with source code. An automated vulnerability discovery system based on tokenized Python code, trained on a deep neural network is implemented by Sultan and El Sayed [7]. The architecture combines sequential modeling with dense layers and is benchmarked on datasets. They demonstrate high detection accuracy for known vulnerability types, which proves that deep learning method is able to be applied in static code analysis.

Seas et al. [8] present a technique for automatic source code threat detection employing on deep representation learning. To achieve this, they encode the source code into feature vectors, and pipe them with the help of classification layers. On a multi-label dataset, the model performs well in detecting multiple vulnerability types at the same time and with generalization capability. Jeon and Kim [9] AutoVAS, based on deep learning and static code features end-to- end vulnerability analysis system. Semantic code patterns are extracted by AutoVAS with GRUs and are fed to a detection layer. We evaluate it on a number of open-source repositories and demonstrate its high accuracy in vulnerability function classification. Brown et al. [10] investigated the application of AutoML for improving deep learning approaches in malware detection. While specializing in malware, their pipeline includes the automation of hyperparameters tuning, features selection, and model selection, leading the faster model training for cybersecurity use cases. Farasat and Posegga [11] discuss on learning vulnerabilities in Python source code. With n-gram features and transformer-based models, they were able to achieve high performance in finding unsafe coding patterns. This paper demonstrates the importance of language dependent features crafting.

In [12] Mohamed provides an extensive survey on AI and ML in cyber security for classification and predict the intrusions up to cyber threat intelligence and analysis of software vulnerabilities. The paper talks about Emerging paradigms, including explainable AI and federated learning, and graph-based models. Li et al. [13], a MIRL-based model is proposed for the detection of security vulnerabilities. Their model transforms source code to an intermediate representation (having an option of preserving some basic semantics) for which deep feature extraction is supposedly efficient. Finally, experimental results demonstrate that MIRL can efficiently increase the training speed and enhance vulnerability detection accuracy.Qiqieh et al. [14] presented a smart cyber threat Identification system based on swarm-optimized machine learning. The model utilizes a swarm intelligence methodology to optimize the parameters of some well-known classifiers to improve the anomaly detection performance. The method was tested on a custom dataset and achieved better accuracy and stability than basic methods. It also works well with low computational cost and can adjust to changing attack situations. However, it is limited to only those types of cyber threats which is included in the provided dataset.

Akinloye et al. [15] presented a survey on AI-based threat detection and response systems used to protect critical national networks. The paper reviews different IDS and IPS methods, explaining their strengths and the challenges faced in real-world use. It also explains how AI can help in decision making and automation in fast-changing environments. The authors suggested a new architecture that combines AI, big data, and cloud systems. However, the study does not include any experiments or result comparison to prove how well the proposed idea works. Ghonge et al. [16] published a book chapter on Cybersecurity and Digital Forensics and highlighted emerging issues namely data explosion, device heterogeneity and sophisticated threat terrains. The chapter argues to implement AI and machine learning to handle large data logs and accelerating the forensic analysis. It also indicates some future trends and open issues but remains conceptual and does not provide or test models.Dawre et al. the increasing importance of IoT forensics were illuminated [17] and case studies that draw attention to limitations of existing forensic methodologies in the context of smart environments were presented. The authors propose forensic methods for real-time IoT, and anti-counterfeit systems to be linked with AI and enhanced traceability. The study is interesting, but there is no general framework that guarantees that the approach works, as well as a lack of empirical and theoretical validation. Dhanushkodi, Thejas[18] proposed AI based threat detection system with different machine learning techniques to protect the communication systems proactively. The system was properly trained and evaluated using NSL-KDD dataset and demonstrated improved detection and reduced False Positives. The architecture is very flexible for zero-day attacks, and the system applied almost no delay to produce an alarm. However, it has only been tested on a single dataset; so the generalization needs to be proved.

Salem et al. [19] reviewed AI-based cybersecurity detection methods, and classified such techniques into supervised, unsupervised, and reinforcement learning. The paper surveys deep-learning architectures such as CNNs, RNNs and attention mechanism which is used across different datasets which include CICIDS2017 and UNSW-NB15. It is a valuable asset for taxonomies and benchmarking. The review is comprehensive, but lacks implementation results or model comparisons.

An AI-based cyber threat detection model for intelligent engineering systems was developed by Sivakumar et al. [20]. The proposed model combines decision support features with intelligent detection and is inspired by neural engineering and cyber-physical security approaches. The paper focuses on modularity and real-time response, ignores datasets in use, and no experimental results are presented.

Prity et al. [21] presented a model that uses machine learning to detect malware and makes use of explainable AI (XAI) concepts, including SHAP and LIME. The objective of the trained model lies in interpreting malware datasets, and therefore the detection accuracy and interpretability of the proposed method have been improved, compared to that of black-box AI systems. This method helps to reduce the gap between transparency and performance in security applications or software's. However, the study is limited only for malware detection and does not consider a big range of cyber threats.

Lanka et al. [22] developed an AI-based threat detection model that shows honeypot-generated data to identify and block malicious behavior patterns of the threats. Their work describes how traditional ML models trained specifically on simulated attack. Data can improve real time alert systems, especially in terms of early alert warning and anomaly detection. The results are limited because the experiments were done in a controlled honeypot setup. So, it is not clear if the method will work properly in other types of networks. Velasquez et al. [23] proposed a hybrid machine learning method for detecting anomalies in Industry 4.0 systems. They combined different classifiers to improve the accuracy and reduce false alarms. The model was tested on real-time industrial data. However, the authors said that using this model in real systems can be costly and difficult.

In the same way, Ragab et al. [24] developed an AI-based cyber threat detection system using CNN and DBN for Industrial IoT environments. The model performed well on standard datasets and learned features effectively. But it needs high processing power, so it may not work well where systems run on a limited resource.

Paracha et al. [25] presented a conceptual discussion on use of AI models for cyber threat detection in different networks. This paper proposes a taxonomy of methodologies and applications types, with emphasis in the capability of self-learning systems. But it is without the model comparison and model performance measures, which makes it not applicable in practice. Tallam [26] proposed CyberSentinel, a conceptual AI based framework uses advanced techniques such as federated and reinforcement learning as well as adversarial training, for the detection of hidden cyber threats. The paper mentions adaptability for modules but is really just a preprint with no experiments as well as data validation. It's a simply theoretical model, and it needs to be peer- reviewed first and then apply the concepts. Jain A.R. et al. [28] proposed a hybrid RNN–

BiLSTM model for software vulnerability detection using code snippets. The model achieved 95.31% accuracy on the Vulnerability Fix Dataset for SQL Injection and Path Traversal detection. Jain A.R. et al. [29] discussed ML-based cyber-threat detection and proactive mitigation. They highlighted challenges such as data imbalance, interpretability, and adversarial attacks.

II.1 Related Challenges

The development and deployment of cyber threat detection systems faces so many practical challenges:

1. Data Quality and Availability: Finding high quality datasets includes proper directions for training and testing is the primary issue. In most of the time, open source vulnerabilities are not properly identified just because of poor quality of the dataset, which can seriously affect performance of the model.

2. Complexity of Cyber Threats: The nature and behavior of cyber attacks increase day by day. So, accurately classifying the cyber vulnerabilities is a challenging and time-consuming task.

3. Model Creation: It is really very important that the model generalizes in a proper manner and direction so that it detects hidden vulnerabilities to maintain the high detection accuracy.

4. Integration with Existing Systems: Integrating the proposed model into existing cybersecurity platforms may not be straightforward due to large compatibility issues and other potential performance overhead.

III.PROPOSED METHODOLOGY

We propose a deep learning based hybrid model for automated classification and detection of hidden vulnerabilities in software source code or open source libraries. The method combines sequential learning functionality with global features to enrich vulnerability classification and prediction to detect attacks like Cross-Site Scripting (XSS) and Command Injection. The whole pipeline is divided into four main categories: data preprocessing, vector generation, hybrid model development and classification.

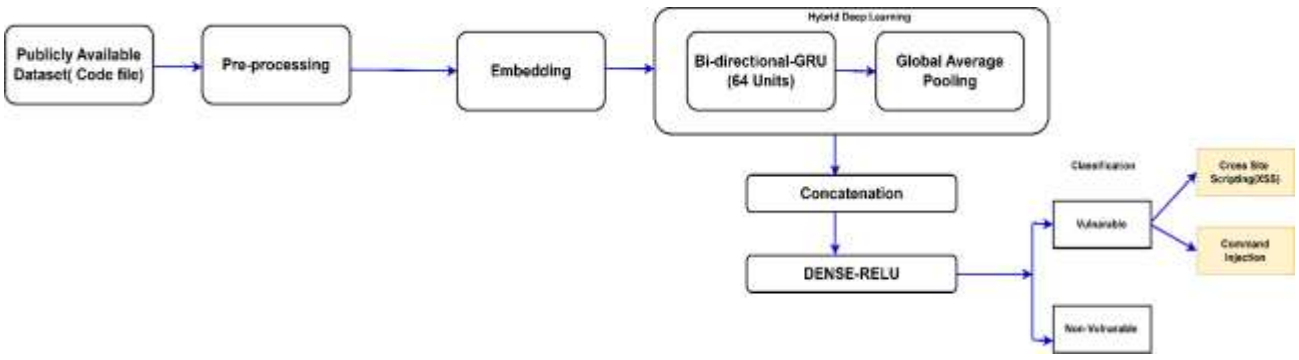

Figure 1: Architecture of the Proposed Hybrid Deep Learning Framework

Source: Authors (2026).

III.1 Data Collection and Preprocessing

For data collection and preprocessing we use publicly available dataset named Vulnerable and Fixed Code dataset from Kaggle [27]. The dataset contains source code files collected from real-world scenarios. All files are labeled based on the type of cyber threats, which helps in training and evaluating the model properly.

Preprocessing technique contains:

- Duplicate Removal: to eliminate duplicate code samples and prevent any type of data leakage.
- Feature Engineering: where raw input is transformed into structured formats such as fixed size tokens.

- Descriptive Statistics: in this, some extra features such as token counts and function call frequencies were extracted which provides richer input for the model training.

III.2 Embedding Layer

The output from preprocessing layer is converted into numerical vectors using an embedding layer. This process converts the code into proper dense vectors which is suitable for neural processing so it preserves its semantic structure.

III.3 Hybrid Deep Learning Architecture

To capture sequential as well as contextual information in source code sequences, the proposed automated deep learning model integrates Bidirectional GRU, GAP and dense layers with ReLU activation. The mathematical formulation of each and every component is given below.

III.3.1 Input Representation and Embedding

The source code can be represented as a no. of tokens.

$$X = \{x1, x2, \ldots\ldots\ldots\ldots, xT\} \quad (1)$$

where T is the sequence length and for each token $xt \epsilon Rd$ is converted to an embedding vector using an embedding matrix $E \in R^{|V|*d}$, where |V| is the vocabulary size and d is the embedding dimension.

$$et = (xt), \forall t \epsilon [1, T] \quad (2)$$

$$\Rightarrow (X) = \{e1, e2, \ldots\ldots\ldots\ldots\ldots, eT\}, \quad et \epsilon Rd\ ] \quad (3)$$

III.3.2 Bidirectional GRU Layer (BiGRU)

The embedded sequence is then passed through a Bidirectional GRU, which captures and learns both forward and backward dependencies.

Let:

$\overrightarrow{h_t}$ be the forward GRU hidden state at time t

$\overleftarrow{h_t}$ be the backward GRU hidden state at time t

The output of the BiGRU at time step t is achieved by concatenating the past hidden state and the future hidden state:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}\ ] \epsilon R^{2h} \quad (4)$$

where,

h=number of hidden units (here h=64)

The full sequence of BiGRU outputs is:

$$H = \{h1, h2, \ldots\ldots\ldots\ldots\ldots, hT\}\ H \epsilon RT*2h \quad (5)$$

III.3.3 Global Average Pooling (GAP)

Global Average Pooling computes the average of hidden states across each time steps, and creates a fixed-length representation that shows the entire sequence:

$$g = (1/T) \sum_{t=1}^{T} h_t,\ g \in R^{2h} \quad (6)$$

As a result, a fixed-length vector *g* is produced, capturing the overall information of the input sequence.

III.3.4 Concatenation Layer

If additional processed representations or auxiliary inputs (e.g., statistical features or secondary embeddings) are used, then they can be concatenated with g.

Assuming $z \in R^k$ is an auxiliary feature vector, the concatenated output is:

$$c = [g; z]\ R2h+k \quad (7)$$

In the absence of any additional input, the context vector c is taken to be equal to g.

III. 3.5 Dense + ReLU Layer

The combined vector is then given to a fully connected dense layer with ReLU (Rectified Linear Unit):

$$f = ReL(W_1 c + b_1) \in R^{d1} \quad (8)$$

where:

$W_1 \in R^{d1*(2h+K)}$

$b_1 \in R^{d1}$

d_1 = no. of units in dense layer

Dense layer is defined as:

$$ReL(x) = max(0, x) \quad (9)$$

III.3.6 Output Layer (Binary or Multi-Class Classification)

Finally, the model performs classification using a SoftMax activation and it totally depends on the task:

Binary classification (Vulnerable vs Non-Vulnerable):

$$\hat{y} = \sigma(W_2 f + b_2),\ \sigma(x) = 1 / (1 + e^{-x}) \quad (10)$$

Multi-class classification (XSS, Command Injection):

$$\hat{y}_j = \exp((W_2 f + b_2)_j) / \sum_{k=1}^{c} \exp((W_2 f + b_2)_k) \quad (11)$$

where:

$W_2 \in R^{C*d1}$

$b_2 \in R^{C}$

C = no. of classes (3: XSS, Command Injection, Non-Vulnerable)

As shown in Figure 1, the architecture is designed to capture sequential patterns and aggregate features, enabling the detection of both structural and contextual anomalies in source code / snippets.

III.4 Pseudocode for your proposed hybrid deep learning architecture:

Input: Source code dataset D = {x_1, x_2, ..., x_n} with labels L = {vulnerable, non-vulnerable}

Output: Predicted labels $\hat{Y}$ = {$\hat{y}_1$, $\hat{y}_2$, ..., $\hat{y}_n$}

1: Initialize BiGRU model with 64 hidden units

2: Initialize Embedding layer, Global Average Pooling layer, Dense-ReLU layer, and Output layer

// Preprocessing Phase

3: for each source code file x in D do

4:Remove duplicate or redundant code samples

5:Tokenize x and perform feature extraction (e.g., syntax trees, statistics)

6:Convert to embedding vectors e = Embed(x)

7:Append e to the list of embedded inputs E 8: end for

// Model Forward Pass

9: for each embedded sequence e in E do

10:h_forward ← GRU_forward(e)

11:h_backward ← GRU_backward(e)

12:h_combined ← Concatenate(h_forward, h_backward)

13:g ← GlobalAveragePooling(h_combined)

14:if auxiliary features z exist then

15:c ← Concatenate(g, z)

16:else

17:c ← g

18:end if

19: f ← ReLU($W_1 \cdot c + b_1$) // Dense layer with ReLU

20: ŷ ← Softmax($W_2 \cdot f + b_2$) // or Sigmoid for binary

21:Append ŷ to Ŷ

22: end for

23: Return predicted labels Ŷ

IV. RESULTS AND DISCUSSIONS

IV.1 Experimental Setup

The hybrid deep model was developed in Python 3.10 and relying on basic machine learning and data processing libraries such as TensorFlow, Keras, Pandas, and matplotlib. The development setting was based on Windows 11 operating system with an Intel(R) Core i7-processor (3.10 GHz), 64 GB-RAM, and an NVIDIA GeForce RTX 3050 GPU. This configuration allowed us to efficiently train the model, perform fast matrix computations, and visualize the model during model evaluation

IV.2 Dataset and Preprocessing

We used the experimental data set Vulnerable and Fixed-Code Dataset [28], available at Kaggle, which has labeled multilingual code snippets of vulnerabilities from real-world. Besides the dataset, we programmatically collected open-source repositories, including Keras, TheAlgorithms/Python, and Requests, to collect more Python source code with the pydriller library. The below script was used to clone the repositories and extract. py files from commit history:

import os import git

from pydriller import RepositoryMining

repos = [

"https://github.com/keras-team/keras", "https://github.com/TheAlgorithms/Python", "https://github.com/kennethreitz/requests",

]

clone_dir = "repos" os.makedirs('w2v', exist_ok=True) pythontraining = ""

for repo_url in repos:

repo_name = repo_url.split('/')[-1]

repo_path = os.path.join(clone_dir, repo_name)

if not os.path.exists(repo_path): git.Repo.clone_from(repo_url, repo_path)

for commit in RepositoryMining(repo_path).traverse_commits(): for m in commit.modifications:

filename = m.new_path

if filename and filename.endswith(".py"): code = m.source_code

if code:

pythontraining += "\n\n" + code

with open('w2v/pythontraining.txt', 'w', encoding='utf-8') as outfile: outfile.write(pythontraining)

This preprocessing step ensured a clean and enriched code corpus for potential embedding training or comparative analysis.

IV.3 Model Training and Evaluation

The hybrid architecture consisting of Bidirectional GRU (64 units) followed by Global Average Pooling, Dense- ReLU, and classification head was trained via categorical cross-entropy loss and Adam optimiser. Code embeddings were fed into the model in the form of numeric vectors which incorporated syntactic and semantic token information.

The model was evaluated using standard classification metrics:

1. Accuracy: The ratio of correctly predicted instances to the total instances. It provides a general measure of the model's performance.

2. Precision: The ratio of correctly predicted positive observations to the total predicted positives. It indicates how many of the predicted positive instances were actually correct.

3. Recall (Sensitivity): The ratio of correctly predicted positive observations to all observations in actual class.

4. F1 Score: The weighted average of Precision and Recall, providing a balance between the two. It is particularly useful when the class distribution is imbalanced.

F1 Score = 2 ∗ (Precision × Recall)/Precision+Recall (12)

Training and validation were performed at 80% and 20% split, respectively. The classification aimed to identify vulnerable code and classify it as either Site Scripting (XSS) or Command Injection and further categorized non- vulnerable code segment.

The bar charts below compare the performance of five different models — CNN, LSTM, GRU, BiGRU, and the Proposed Hybrid Model across four key metrics: Accuracy, Precision, Recall, and F1 Score.

Natural Resources for Human Health
eISSN: 2583-1194

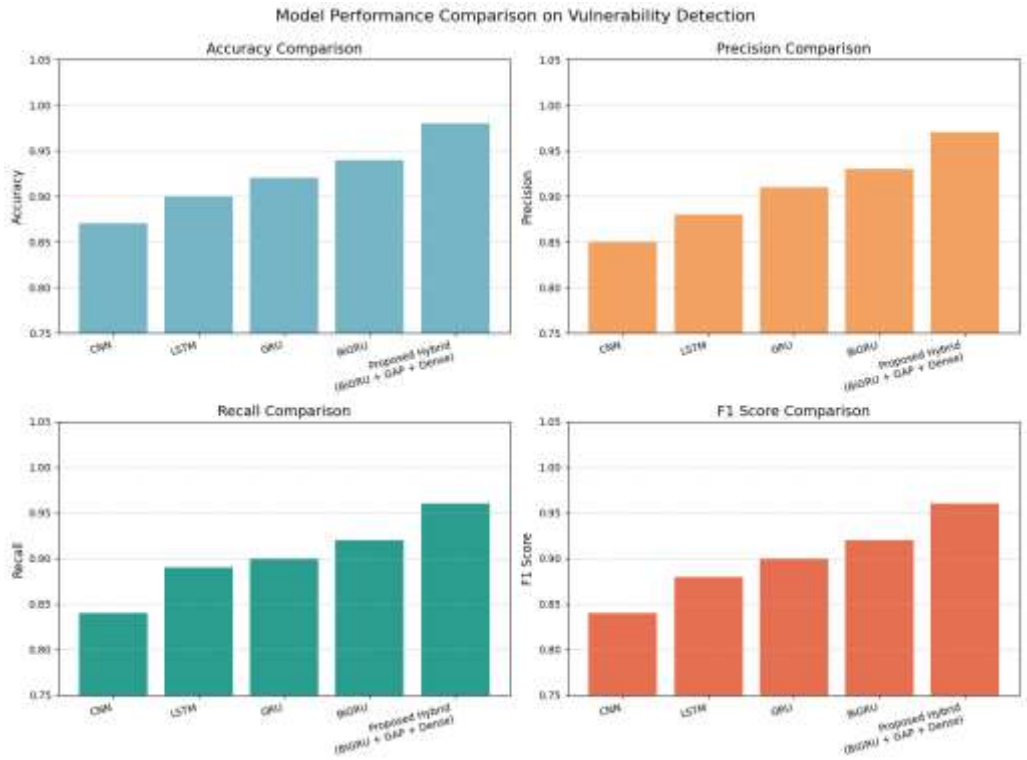

Figure 2: Model Performance Comparison on vulnerability detection

Source: Authors (2026).

Each chart shows how well the models perform in detecting software vulnerabilities:

- The Proposed Hybrid Model consistently has the highest bars, showing the best performance across all metrics.
- BiGRU also performs strongly, followed by GRU and LSTM.
- CNN has the lowest values among the five models.

These charts make it easy to see that combining BiGRU with Global Average Pooling and Dense layers leads to the most accurate and reliable results.

The following table compares different deep learning models used for detecting vulnerabilities in source code. Each model is evaluated based on the above four important metrics:

Table 1- Performance Analysis of Baseline and Proposed Hybrid Models Using Standard Evaluation Metrics

Model	Accuracy	Precision	Recall	F1 Score
CNN	0.87	0.85	0.84	0.84
LSTM	0.9	0.88	0.89	0.88
GRU	0.92	0.91	0.9	0.9
BiGRU	0.94	0.93	0.92	0.92
Proposed Hybrid (BiGRU + GAP + Dense)	**0.98**	**0.97**	**0.96**	**0.96**

Source: Authors (2026).

From the comparison:

- CNN showed basic performance with 87% accuracy.
- LSTM and GRU performed better, with GRU reaching 92% accuracy.
- BiGRU gave even better results by understanding both forward and backward sequences.
- The Proposed Hybrid Model (BiGRU + GAP + Dense) performed the best, with 98% accuracy, making it highly effective in detecting and classifying security vulnerabilities.

This clearly shows that combining multiple layers and techniques improves model performance.

IV.4 Result Analysis

Our preliminary results indicated that the model performance reached 98% accuracy, indicating the good performance of employing BiGRU and pooling operations for sequence modeling and the task of vulnerability discovery. A higher precision in detection confidence was observed for vulnerable cases than for non-vulnerable cases, showing better feature generalization for challenging attack structures.

V. CONCLUSION

In this paper, a hybrid deep learning model is proposed using Bidirectional GRU, Global Average Pooling, and dense layers to detect and classify source code vulnerabilities. The aim is to meet the growing need for smart cybersecurity systems that can find important as well as major threats such as XSS and command injection. The model was trained using a dataset that includes multiple programming languages and real world code from different repositories. This helps the system learn different types of vulnerabilities from real world examples. The results indicate that the model can understand both the structure and it's the meaning of the code, which helps it correctly identify vulnerable and non vulnerable code. The use of BiGRU along with GAP helps the model detect complex attack patterns in the code. The model works significantly and handles large data on modern real world systems. Overall, the experimental results support the use of hybrid deep learning models for detecting threats and proactive defense, especially in tools that help software developers write secure code.

In the future, the work can be extended by adding more types of vulnerabilities, using attention mechanisms to better understand the model decisions, and applying the model to more programming languages and different development environments.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Prasad A.Lahare, Manoj A. Wakchaure

Methodology: Prasad A.Lahare, Manoj A. Wakchaure

Investigation: Prasad A.Lahare, Manoj A. Wakchaure.

Discussion of results: Prasad A.Lahare, Manoj A. Wakchaure

Writing – Original Draft: Prasad A.Lahare

Writing – Review and Editing: Prasad A.Lahare, Manoj A. Wakchaure

Resources: Prasad A.Lahare, Manoj A. Wakchaure

Supervision: Prasad A.Lahare, Manoj A. Wakchaure.

Approval of the final text: Prasad A.Lahare, Manoj A. Wakchaure

REFERENCES

[1] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," in Proc. IEEE Int. Conf. Smart Computing (SMARTCOMP), Hong Kong, China, 2017, pp. 1–8. doi: 10.1109/SMARTCOMP.2017.7946998.

[2] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an Intrusion Detection System Using a Filter-Based Feature Selection Algorithm," IEEE Transactions on Computers, vol. 65, no. 10, pp. 2986–2998, Oct. 2016. doi: 10.1109/TC.2016.2519914.

[3] N. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A Deep Learning Approach for Network Intrusion Detection System," EAI Endorsed Transactions on Security and Safety, vol. 3, no. 9, 2016. doi: 10.4108/eai.3-12-2015.2262516.

[4] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end Encrypted Traffic Classification with One-Dimensional Convolution Neural Networks," in Proc. IEEE Int. Conf. Intelligence and Security Informatics (ISI), Beijing, China, 2017, pp. 43–48. doi: 10.1109/ISI.2017.8004872.

[5] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang, "Software Vulnerability Analysis and Discovery Using Deep Learning Techniques: A Survey," IEEE Access, vol. 8, pp. 197158–197172, 2020. doi: 10.1109/ACCESS.2020.3034766.

[6] S. M. Arıkan, A. Koçak, and M. Alkan, "Automating Shareable Cyber Threat Intelligence Production for Closed Source Software Vulnerabilities: A Deep Learning Based Detection System," International Journal of Information Security, vol. 23, pp. 3135–3151, 2024. doi: 10.1007/s10207-024-00882-4.

[7] M. T. Sultan and H. El Sayed, "A Deep Learning-Based Approach for an Automated Vulnerability Detection in Source Code," in Proc. IEEE Global Conf. Artificial Intelligence and Internet of Things (GCAIoT), Dubai, United Arab Emirates, 2023, pp. 17–22. doi: 10.1109/GCAIoT61060.2023.10385129.

[8] C. Seas, G. Fitzpatrick, J. A. Hamilton, and M. C. Carlisle, "Automated Vulnerability Detection in Source Code Using Deep Representation Learning," in Proc. IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2024, pp. 484–490. doi: 10.1109/CCWC60891.2024.10427574.

[9] S. Jeon and H. K. Kim, "AutoVAS: An Automated Vulnerability Analysis System with a Deep Learning Approach," Computers & Security, vol. 106, Art. no. 102308, 2021. doi: 10.1016/j.cose.2021.102308.

A. Brown, M. Gupta, and M. Abdelsalam, "Automated Machine Learning for Deep Learning Based Malware Detection," Computers & Security, vol. 137, Art. no. 103582, 2024. doi: 10.1016/j.cose.2023.103582.

[10] T. Farasat and J. Posegga, "Machine Learning Techniques for Python Source Code Vulnerability Detection," in Proc. 14th ACM Conf. Data and Application Security and Privacy (CODASPY), 2024, pp. 151–153. doi: 10.1145/3626232.3658637.

[11] N. Mohamed, "Artificial Intelligence and Machine Learning in Cybersecurity: A Deep Dive into State-of-the-Art Techniques and Future Paradigms," Knowledge and Information Systems, 2025. doi: 10.1007/s10115-025-02429-y.

[12] X. Li, L. Wang, Y. Xin, Y. Yang, and Y. Chen, "Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning," Applied Sciences, vol. 10, no. 5, Art. no. 1692, 2020. doi: 10.3390/app10051692.

[13] O. Qiqieh, J. Alzubi, J. Alzubi, K. C. Sreedhar, and A. M. Al-Zoubi, "An Intelligent Cyber Threat Detection: A Swarm-Optimized Machine Learning Approach," Alexandria Engineering Journal, vol. 115, pp. 553–563, 2025. doi: 10.1016/j.aej.2024.12.039.

A. Akinloye, S. Anwansedo, and O. T. Akinwande, "AI-Driven Threat Detection and Response Systems for Secure National Infrastructure Networks: A Comprehensive Review," International Journal of Latest Technology in Engineering, Management & Applied Science, vol. 13, no. 7, pp. 82–92, 2024. doi: 10.51583/IJLTEMAS.2024.130710.

[14] M. M. Ghonge, S. Pramanik, R. S. Mangrulkar, and D.-N. Le, Cybersecurity and Digital Forensics: Challenges and Future Trends. Hoboken, NJ, USA: John Wiley & Sons, 2022.

[15] Dawre, I. Kheria, R. S. Mangrulkar, and M. M. Ghonge, "The Need for Internet of Things (IoT) Forensics: Case Studies and Analysis," in Cyber Security Threats and Challenges Facing Human Life. Boca Raton, FL, USA: Chapman and Hall/CRC, 2022, pp. 81–96.

[16] Dhanushkodi and S. Thejas, "AI Enabled Threat Detection: Leveraging Artificial Intelligence for Advanced Security and Cyber Threat Mitigation," IEEE Access, vol. 12, pp. 173127–173136, 2024. doi: 10.1109/ACCESS.2024.3493957.

A. H. Salem, S. M. Azzam, O. E. Emam et al., "Advancing Cybersecurity: A Comprehensive Review of AI-Driven Detection Techniques," Journal of Big Data, vol. 11, Art. no. 105, 2024. doi: 10.1186/s40537-024-00957-y.

[17] Sivakumar, N. R. Salman, F. R. Salman, H. R. Salimova, and E. Ghimire, "AI-Driven Cyber Threat Detection: Enhancing Security Through Intelligent Engineering Systems," Journal of Information Systems Engineering and Management, vol. 10, no. 19s, 2025.

[18] F. S. Prity, M. S. Islam, E. H. Fahim et al., "Machine Learning-Based Cyber Threat Detection: An Approach to Malware Detection and Security with Explainable AI Insights," Human-Intelligent Systems Integration, vol. 6, pp. 61–90, 2024. doi: 10.1007/s42454-024-00055-7.

[19] P. Lanka, K. Gupta, and C. Varol, "Intelligent Threat Detection—AI-Driven Analysis of Honeypot Data to Counter Cyber Threats," Electronics, vol. 13, no. 13, Art. no. 2465, 2024. doi: 10.3390/electronics13132465.

[20] D. Velasquez, E. Perez, X. Oregui, A. Artetxe, J. Manteca, J. E. Mansilla, M. Toro, M. Maiza, and B. Sierra, "A Hybrid Machine-Learning Ensemble for Anomaly Detection in Real-Time Industry 4.0 Systems," IEEE Access, vol. 10, pp. 72024–72036, 2022.

[21] Ragab, M. Basheri, N. N. Albogami, A. Subahi, O. A. Abdulkader, H. Alaidaros, H. Mousa, and A. Al-Malaise Al-Ghamdi, "Artificial Intelligence Driven Cyberattack Detection System Using Integration of Deep Belief Network with Convolution Neural Network on Industrial IoT," Alexandria Engineering Journal, vol. 110, pp. 438–450, 2025. doi: 10.1016/j.aej.2024.10.009.

[22] A. Paracha, S. U. Jamil, K. Shahzad, M. A. Khan, and A. Rasheed, "Leveraging AI for Network Threat Detection—A Conceptual Overview," Electronics, vol. 13, no. 23, Art. no. 4611, 2024. doi: 10.3390/electronics13234611.

[23] K. Tallam, "CyberSentinel: An Emergent Threat Detection System for AI Security," arXiv preprint arXiv:2502.14966, 2025

[24] JISCE CSE AIML, "Vulnerability Fix Dataset," Kaggle Dataset. [Online]. Available: https://www.kaggle.com/datasets/jiscecseaiml/vulnerability-fix-dataset. [Accessed: 03-Jun-2026].

[25] Akshay R. Jain, & Atul Agrawal. (2026). A Hybrid RNN-LSTM Framework for Cyber Threat Detection Using Vulnerability Code Snippets. International Journal of Computer Information Systems and Industrial Management Applications, 18(19s), 1096–1106. https://doi.org/10.70917/ijcisim-2026-5103

[26] Jain, A. R., & Agrawal, A. (2026). Cyber threat analysis using machine learning for proactive mitigation: A comprehensive review. Grenze International Journal of Engineering and Technology. (Grenze ID: 01.GIJET.12.1.520).