Computer Science and Information Systems 23(1):397–417 https://doi.org/10.2298/CSIS250314079B

Mitigating Out-of-Vocabulary Challenges in Embedded devices Vulnerability Classification: An Ensemble Embedding Approach with Bidirectional Context Modeling ⋆

Aissa Ben Yahya, Hicham El Akhal, and Abdelbaki El Belrhiti El Alaoui

Computer Science and Applications Laboratory, Faculty of Sciences, Moulay Ismail University of Meknes, B 11201 Zitoune, 50000, Meknes, Morocco
{ai,hi}@edu.umi.ac.ma
a.elbelrhitielalaoui@umi.ac.ma

Abstract. Critical infrastructure is increasingly reliant on embedded systems, which are particularly vulnerable to cyberattacks due to their inherent complexity and interconnectivity. Accurate classification of vulnerabilities in these systems is essential for targeted analysis and mitigation strategies. While pre-trained word embeddings such as Word2Vec, GloVe, and FastText are commonly used for this purpose, their effectiveness is limited by reliance on training corpora that lack domain-specific terminology, leading to challenges with Out-of-Vocabulary words and reduced classification performance. To address this limitation, we propose a novel ensemble embedding technique that combines multiple pre-trained embeddings to improve vulnerability classification in embedded systems. Evaluated on benchmark datasets, including the National Vulnerability Database and the China National Vulnerability Database, our method achieves a 91.50% F1-score on unseen data, outperforming traditional single-embedding approaches. This advancement demonstrates significant potential for enhancing cybersecurity in critical infrastructure applications.

Keywords: ensemble embedding, word embedding, vulnerability classification, critical infrastructure devices security.

1. Introduction

The mass adoption of embedded systems in security-critical areas such as healthcare, transportation, and manufacturing automation requires the protection of such systems in order to make them secure. Owing to the inherently sensitive nature of the data being processed by these systems, coupled with distinctive operational specifications, such systems are highly vulnerable to security breaches. Consequences of such violations have the potential to extend significantly and may include service disruptions [12], improper access to sensitive data [16,4], potential equipment destruction [10], and, in the most severe cases, threatening human life [9].

⋆ This manuscript is an extended version of the paper "Enhanced Classification of Embedded System Vulnerabilities Using Ensemble Embedding and BiLSTM Networks" presented at the 28th International Symposium, IDEAS 2024, Bayonne, France, August 26–29, 2024.

The inherent distinction between embedded system vulnerabilities (ESVs) and general-purpose system vulnerabilities (GPSVs) comes from the unique architectural and operational nature of embedded devices. Unlike servers or desktop computers, embedded systems feature severe restrictions in resources, including limited memory (RAM), processing power (CPU), and power budgets [7]. By virtue of these restrictions, traditional security strategies, such as resource-hungry antivirus software or frequent, large-scale updates[1] , become infeasible. Furthermore, these systems employ domain-specific communication protocols (e.g., CAN bus in the automotive sector, Modbus in industrial controls, and Zigbee in IoT), which often have no built-in, end-to-end security measures inherent in traditional internet protocols. This, in turn, produces quite dissimilar exploit profiles and threat models, where attackers may target physical interfaces[2] (such as JTAG or UART), firmware, or real-time operating systems (RTOS) in ways that are utterly irrelevant in a normal IT environment. This problem is further compounded by the long operational life and the infrequent update schedule of many embedded systems, meaning that weaknesses may persist for years or even decades after discovery. Because of these inherent differences, the current treatment of ESVs and GPSVs under one monolithic category falls short and poses a risk. Without a differentiated scheme of categorization, security professionals cannot accurately prioritize threats relevant to these systems in real-time for analysis. This research, to the best of our knowledge, is the very first to present an automated system of classification that carefully distinguishes ESVs and their GPSV equivalents so that this critically important infrastructure could be secured with much higher precision and efficacy.

Recent advancements in machine learning and deep learning have facilitated the development of automatic classification mechanisms, which have been successfully applied to the realm of vulnerability classification [18,8,6,3,14]. The current state-of-the-art in embedded systems vulnerability classification is represented by the work in [2], which employs Term Frequency Inverse Document Frequency (TF-IDF) to generate weighted word vectors and Support Vector Machines (SVM) as a classifier. However, this approach has limitations, including its reliance on TF-IDF, which overlooks the co-occurrence and similarity relationships between words. Additionally, it lacks a mechanism to handle out-of-vocabulary (OOV) words and relies on SVM, which can be sensitive to feature scaling and fails to capture contextual information, leading to poor generalization to new data. [19] extended this line of research by proposing an algorithm that classified vulnerabilities by associating Common Vulnerabilities and Exposures (CVE) entries with their corresponding CWE-ID. Furthermore, they introduced an improved TF-IDF in [20] that weights words based on their category. Although this algorithm achieved promising results, it has no mechanism of dealing with OOV words.

Traditional machine learning approaches, while effective in general text classification tasks, often face limitations when applied to more specialized contexts. These limitations arise from their reliance on pre-trained word embeddings, which are susceptible to the OOV problem [13]. When words that were not encountered during training appear, they are inadequately represented in the model, resulting in diminished classification performance.

[1] https://www.mottasec.com/article/faq-why-is-embedded-security-so-hard-to-get-right

[2] https://interrupt.memfault.com/blog/diving-into-jtag-part-6

To address these challenges, we introduce a new framework of ensemble embeddings that integrates multiple word embedding approaches, such as GloVe, Word2Vec, and FastText, to have enhanced word representations. This approach markedly minimizes the issues concerning OOV words. Our methodology implements an OOV word fallback approach where OOV word vectors are replaced with those of the most available semantically similar words. This supposes that semantically related words frequently coexist in similar contexts. This approach is implemented in a bidirectional gated recurrent unit (BiGRU) model that characterfully extracts contextual dependency. Also included in our research is the creation of a benchmark dataset that was synthesized from multiple databases, presenting a versatile and representative set of data that could function equally well both to train and test the model. Our research's contributions consist of:

- A new methodology to differentiate and categorize vulnerability reports so that those of embedded systems could be differentiated from general-purpose systems to allow concentrated research.
- A new ensemble embedding scheme that simultaneously uses several schemes of word embeddings with a dynamically determined threshold to reduce the effect of OOV words.
- A Benchmark data set, which is compiled from several sources, presents a diverse and representative collection of data that helps in real-world deployment assessment.

This paper is structured as follows: we provide an overview of relevant literature in section 2, followed by a descriptive account of our methodology in section 3. section 4 consists of the results of the assessment in reference to our proposed mechanism. section 5 concludes the paper with a summary of our primary findings and conclusions from our research.

2. Related Work

Software vulnerability classification has also seen tremendous breakthroughs with the use of ML and DL methods. Current text representation methods have been largely grouped into two classes: (1) conventional feature engineering approaches, and (2) encoding with the help of word embedding schemes. Below, we critically analyze these methods, highlight their limitations, and explain how they motivate our proposed approach.

2.1. Traditional Feature Engineering Methods

Earlier works concentrated almost entirely on term-frequency-based text representation and dimension reduction methods. Specifically, [21] employed a document-term matrix (DTM) alongside chi-square feature selection to treat high-dimensional data. Likewise, [15] advanced a multi-label prediction model that used DTM-based word representation and weighting to attribute vulnerability characteristics. Although these approaches perform well in dealing with simple term distributions, they lack semantic relationships or contextual importance of words, hence yielding suboptimal performance when applied in texts with specialized domains. Likewise, [6] integrated TF-IDF alongside information gain (IG) and deep neural networks (DNNs). However, TF-IDF ignores semantic and local contextual knowledge, and IG focuses on worldwide discriminative features at the

expense of specialized ones in the given domains. Such drawbacks impair them in separating fine-grained categories of vulnerabilities. Extending this paradigm, [1] presented ThreatZoom, a framework that associates Common Vulnerabilities and Exposures (CVEs) with Common Weakness Enumerations (CWEs) based on n-gram features with TF-IDF weights. Though local sequences of words have been preserved with n-grams, the scheme borrows all the inherent flaws in the TF-IDF weight assignment scheme. Emphasizing the weaknesses of global term weights in more detail, [2] used Support Vector Machines (SVM) with TF-IDF to extract embedded system vulnerabilities. Although TF-IDF successfully downweights popular terms (e.g., "vulnerability" or "system"), its failure to integrate word co-occurrences and similarity lowers precision in multi-domains. This necessitates a process that inserts semantic and contextual knowledge into the word vectors, overcoming traditional weaknesses.

2.2. Word Embedding Techniques

To overcome the semantic shortcomings of TF-IDF, alternative strategies have been investigated by several studies. For example, [14] incorporated pre-trained GloVe embeddings and Convolutional Neural Networks (CNNs). Their approach, however, suffers from OOV words since GloVe has been pretrained on a static corpus and lacks adaptability to adapt to the emerging new terms. Likewise, [5] incorporated pre-trained Word2Vec embeddings and CNNs to learn sentence contexts. Although this model enhanced semantic understanding, it still had issues with OOV words since it also uses static pre-trained embeddings. Taking it a step further, [17] proposed a self-attention-based Deep Neural Network (SA-DNN) that seeks to emphasize important words in vulnerability descriptions. Nonetheless, self-attention mechanisms are prone to noise, which results in the model emphasizing background or irrelevant words, hence suffering in terms of robustness. Despite these efforts that have gone into building vulnerability classifiers, these works have been constrained by the failure to adopt a good OOV handling scheme, hence limiting software vulnerability classification. With these weaknesses in mind, a more adaptive and inclusive solution that minimizes OOV word influence and achieves contextual and semantic understanding on a good note becomes important.

2.3. Research Gaps and Motivation

Classification approaches suffer severely from the weaknesses in applying word embedding approaches, which in turn also affect solving the OOV word problem. There always remain two fundamental issues: (1) the inefficiency of relying on a single pre-trained approach to capturing the subtle relationship between terms in specialized fields, and (2) the incapability of the existing pre-trained embeddings, such as Word2Vec, GloVe, or FastText, to represent rare appearing or new vulnerability-related terms and thereby suffering from OOV problems. Such weaknesses point to the urgency in developing a new solution that both efficiently deals with OOV words and incorporates both co-occurring and contextual knowledge to attain superior vulnerability classification performance. Our proposed solution exploits these issues by adopting a new ensemble embedding approach in conjunction with a BiGRU model. By combining multiple word embedding models, such as GloVe, Word2Vec, and FastText, our solution produces more descriptive word

representations. This not only reduces OOV-related issues but also augments contextual knowledge and hence produces a stronger and more accurate vulnerability classification solution.

3. The proposed approach

The overall method used to create the embedded system vulnerability classification model is shown in Figure 1. The process has three main phases.

First, in the data collection and preprocessing phase, we gathered a raw dataset from various publicly available sources, including NVD, CNNVD, and the Zero Day Initiative (ZDI). We extracted relevant vulnerability descriptions with a script. Next, we balanced and preprocessed the data to reduce bias and ensure the text was formatted correctly for the model. Then, during the data split and representation phase, we divided the processed data into training, validation, and benchmark sets. To make sure our experiments reflected real-world challenges, we constructed the benchmark set using data unique to each source database and included manually curated, difficult examples. After the split, we applied word embedding to represent the textual data in numerical form. Finally, the Model architecture was established to process the training and validation data. The architecture begins with an Embedding layer, followed by a stack of three Bidirectional LSTM (BiLSTM) layers designed to extract rich contextual information from the descriptions. These are succeeded by two Dense layers. Notably, both the BiLSTM and Dense blocks incorporate Dropout and Batch Normalization to stabilize training and prevent overfitting. The pipeline concludes with a Softmax layer to generate prediction probabilities for the Output classes. The reserved benchmark data is then used to evaluate the performance of the finalized Model.

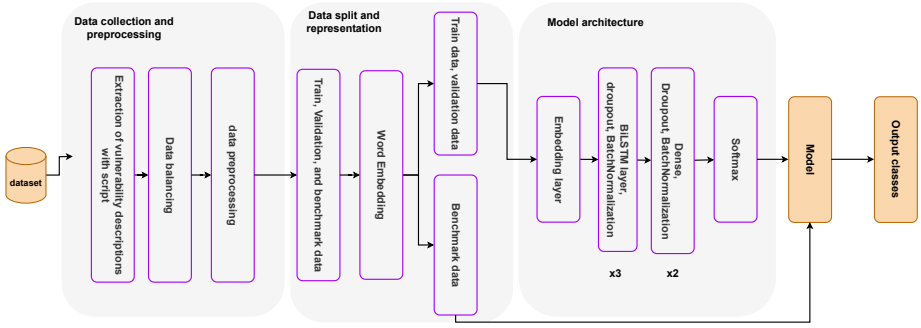

Fig. 1. Main method

3.1. Ensemble Embedding Technique

In natural language processing (NLP), word embedding techniques (WE) such as Word2Vec, GloVe, and FastText are widely used to map words into dense vector representations, cap-

turing syntactic and semantic relationships. However, a critical limitation of these methods is their susceptibility to OOV words, which are typically assigned null vectors if absent from the training corpus. This issue is particularly severe in domain-specific classification tasks, such as software vulnerability classification, where rare or emerging terminology is important for better classification performance. OOV words degrade model generalizability and robustness, as their misrepresentation introduces noise into tasks like classification.

To tackle this challenge, we suggest an ensemble embedding method that combines multiple word embeddings to lessen the impact of out-of-vocabulary words on performance. Our approach is based on two main ideas: (1) Different word embedding techniques trained on various datasets will capture more words, and (2) Similar words in the vector space will probably have similar meanings and functions in the text. Unlike typical methods that depend on a single source of word embeddings, our strategy uses a hierarchical structure. A primary word embedding acts as the main representation, while secondary embeddings provide backup options.

To illustrate the limitations of single WE approaches, consider the CVE in Figure 2 describing a vulnerability in the IEEE 802.15.4 Zigbee protocol. Critical terms such as *802.15.4, Zigbee, coordinator*, and *PHY* are domain-specific and absent in Word2Vec, as shown in Table 1. Traditional methods relying on a single WE would assign a zero vector to these OOV words, degrading classification accuracy.

Our proposed ensemble embedding technique implements the fallback mechanism to reduce the effect of these OOV words. The steps are as follows:

Our proposed technique implements an intelligent fallback mechanism. When a word is not found in the primary embedding, the algorithm searches for it in the secondary embeddings. If found, it retrieves the top-10 most similar words from that secondary model. However, simply choosing the first available substitute is not optimal. To ensure only the most relevant substitutions are made, we introduce a dynamic similarity threshold. For example, if the OOV word `Zigbee` is found in GloVe or FastText, our algorithm analyzes its top-10 neighbors as shown in Table 2, calculates an appropriate threshold (see subsection 3.2) for this specific word, and selects the best candidate. In our example, the dynamic threshold for the GloVe neighborhood of ”zigbee” is approximately 0.386; therefore, the following words have scores above this threshold and exist in the primary (Word2Vec) embedding: `hdcp (Score: 0.44)` and `aminoglycosides (Score: 0.41)`, the one with higher similarity score and exist in primary WE will be selected and use as a substitute for the missing word.

CVE-2024-3043

An unauthenticated IEEE 802.15.4 'co-ordinator realignment' packet can be used to force Zigbee nodes to change their network identifier (pan ID), leading to a denial of service. This packet type is not useful in production and should be used only for PHY qualification.

Fig. 2. A CVE describing a vulnerability in the IEEE 802.15.4 Zigbee protocol

Table 1. Comparison of embedded devices vulnerability word Coverage

Words	Word2vec	Glove	Fasttext
unauthenticated	✓	✓	✓
IEEE	✓	✓	✓
802.15.4	✗	✓	✓
co-ordinator	✗	✓	✓
realignment	✓	✓	✓
Zigbee	✗	✓	✓
nodes	✓	✓	✓
identifier	✓	✓	✓
denial	✓	✓	✓
packet	✓	✓	✓
production	✓	✓	✓
PHY	✗	✓	✓
qualification	✓	✓	✓

Table 2. Top similar words to *zigbee* in secondary embeddings (GloVe and FastText). Substitutes marked with (✓) exist in the primary embedding (Word2Vec).

GloVe			**FastText**		
Word	**Similarity**	**In Primary**	**Word**	**Similarity**	**In Primary**
802.15.4	0.58	✗	Zigbee	0.82	✗
z-wave	0.47	✗	Z-Wave	0.68	✗
hdcp	0.44	✓	ZigBee	0.66	✗
homeplug	0.44	✗	**DeviceNet**	0.65	✓
upnp	0.42	✗	1-Wire	0.63	✗
wml	0.41	✗	**Modbus**	0.63	✓
nmea	0.41	✗	**HomePlug**	0.62	✓
aminoglycosides	0.41	✓	**ecobee**	0.61	✓
passivhaus	0.40	✗	AllJoyn	0.61	✗
displayport	0.40	✗	**BACnet**	0.60	✓

3.2. Dynamic Threshold Calculation

A key innovation of our approach is adapting a dynamic threshold that is based on the local statistics of a word's semantic neighborhood. The process is as follows:

1. For an OOV word found in a secondary embedding, retrieve the top-10 most similar words and their cosine similarity scores.
2. Calculate the mean (μ) and standard deviation (σ) of these 10 scores. The mean represents the average similarity of the neighborhood, while the standard deviation measures its dispersion.

3. The dynamic threshold for this specific word is then set to $T = \mu - \sigma$. Where T is the threshold.

This data-driven threshold is strict for words with tight, high-similarity neighborhoods and more forgiving for words with looser neighborhoods. This ensures that substitutions are always contextually appropriate, maximizing the relevance of the fallback mechanism. The algorithm 1 formalizes the complete procedure.

4. Evaluation

4.1. Data collection

The dataset utilized in this research was assembled from various esteemed sources of vulnerability reports, which include the National Vulnerability Database (NVD) [3], the Chinese National Vulnerability Database (CNNVD) [4], the China National Vulnerability Database (CNVD) [5], VarIoT [6], the Cyber Emergency Response Team (ICS-CERT) [7], Fortiguard [8], and the Zero Day Initiative (ZDI) [9]. The aforementioned sources contributed a heterogeneous and comprehensive array of software vulnerability reports, thus facilitating the development of a robust dataset [10] aimed at the classification of vulnerabilities in embedded systems.

In preparing to get the NVD data, we downloaded JSON files from the NVD feeds [11]. We, however, excluded those entries with a flag that had been set to RESERVED, REJECTED, or DISPUTED since they are unconfirmed vulnerabilities and could not form part of training our models.

We extracted data in XML format from both CNNVD's and CNVD's official websites. Since the descriptions and corresponding info were in Chinese, all entries were translated into English through the Google Translate API [12] to ensure consistency in our dataset.

Due to the absence of a direct download feature in VarIoT, we created a web scraper utilizing Python to retrieve the required data. Meanwhile, data obtained from ICS-CERT, FortiGuard, and ZDI were used exclusively for testing and were excluded from our training dataset. In this study, these external sources are referred to collectively as "Unseen." To ensure the maintenance of high standards of data quality and relevance, we undertook a manual curation of their data to ensure its alignment with our research objectives.

A comprehensive overview of the collected dataset is presented in Table 3.

4.2. Data preprocessing

To prepare the data for modeling, we applied the following preprocessing steps:

[3] https://nvd.nist.gov/
[4] https://www.cnnvd.org.cn/
[5] https://www.cnvd.org.cn/
[6] https://www.variotdbs.pl/vulns/
[7] https://www.cisa.gov/
[8] https://www.fortiguard.com/
[9] https://www.zerodayinitiative.com/
[10] https://www.kaggle.com/datasets/aissaultimate/ensemble-embedding-data
[11] https://nvd.nist.gov/vuln/data-feeds
[12] https://pypi.org/project/googletrans/

Algorithm 1 Ensemble Embedding Approach

Input: Set of word embeddings models $E = \{E_1, E_2, \ldots, E_n\}$, where E_1 is the primary embedding model.
Input: Vocabulary V of words to be embedded.
Input: Vector dimension d.
Output: Word vector matrix M of size $|V| \times d$.
for each word $w \in V$ **do**
    $is\_assigned \leftarrow$ false
    ▷ **Step 1: Primary Embedding Search**
    if $w \in E_1$ **then**
        $v_w \leftarrow E_1(w)$
        $is\_assigned \leftarrow$ true
    end if
    if not $is\_assigned$ **then**
        ▷ **Step 2: Secondary Embedding Fallback**
        for each secondary embedding $E_i \in \{E_2, \ldots, E_n\}$ **do**
            if $w \in E_i$ **then**
                Let S be the list of (word, score) pairs from the top-10 most similar words to w in E_i.
                if S is not empty **then**
                    Let $Scores$ be the list of scores from S.
                    $\mu \leftarrow$ mean$(Scores)$
                    $\sigma \leftarrow$ standard_deviation$(Scores)$
                    $dynamic\_threshold \leftarrow \mu - \sigma$
                    for each $(s_j, score_j) \in S$ **do**
                        if $score_j \geq dynamic\_threshold$ and $s_j \in E_1$ **then**
                            $v_w \leftarrow E_1(s_j)$
                            $is\_assigned \leftarrow$ true
                            break ▷ Break from inner loop (substitutes)
                        end if
                    end for
                end if
            end if
            if $is\_assigned$ **then**
                break ▷ Break from outer loop (embeddings)
            end if
        end for
    end if
    if not $is\_assigned$ **then**
        ▷ **Step 3: Final Subword Fallback**
        ▷ Assumes E_1 is a subword-capable model like FastText.
        $v_w \leftarrow$ generate_subword_vector(E_1, w)
    end if
    Store v_w in matrix M.
end for
return Matrix M.

– **Lowercase conversion:** We converted all text to lowercase to reduce dimensionality and improve model performance.

Table 3. Data set collected from different databases

	ESV	**GPSV**
NVD	13515	32387
CNNVD	12026	27535
CNVD	7457	14364
VarIoT	6775	4586
ICS-CERT, ZDI, Fortiguard (Unseen)	68	100

- **Digit removal:** We removed all digits from the text to focus on the linguistic features of the data.
- **Contraction expansion:** We expanded contractions (e.g., ”you’re” to ”you are”, ”i’m” to ”i am”) to improve tokenization.
- **Stopword removal:** We removed common stopwords, such as ”the”, ”and”, etc., that do not add significant value to the text, to improve model accuracy.
- **Porter Stemming:** We applied Porter Stemming to reduce words to their base form, thereby reducing dimensionality and improving model performance.
- **Special character removal:** We removed special characters from the text to focus on the linguistic features of the data.
- **Base form conversion:** We converted words to their base form (e.g., ”ran” to ”run”) to further reduce dimensionality.

4.3. Data labeling

We utilized a Python script[13] to label the dataset using a two-tiered keyword-based methodology. This script operated with two distinct keyword sets: one specific to embedded systems and another for general-purpose systems. These keyword lists were meticulously compiled through an extensive review of online resources and generative large language models (LLMs), ensuring the inclusion of relevant terms associated with embedded devices, protocols, and software. The same process was applied to construct the keyword list for general-purpose systems. A sample of these keyword sets is provided in Table 4.

The script then processed each vulnerability description, scanning for the presence of keywords from both lists. If a description contained embedded system-related keywords while lacking general-purpose system keywords, it was classified as an embedded system vulnerability. Conversely, if general-purpose system keywords were detected without any embedded system keywords, the vulnerability was categorized accordingly. To reduce ambiguity during model training, entries containing keywords from both lists were omitted from the dataset.

This keyword-based classification strategy was inspired by prior research by Pap et al. [11], which utilized a similar methodology for vulnerability categorization. By adopting this approach, we ensured that our dataset was systematically labeled and optimally structured for model training.

[13] https://github.com/aissa302/CVE-Aanlysis-Project

Table 4. White and black list keywords

Embedded system terms	General-purpose terms
Asus RT	Huawei FusionServer
Huawei Quidway switches	Windows Desktop
Cisco 4000 Series	Apple OS X
Zhone GPON	Xen hypervisor
ZigBee	Google Earth
Modbus	Dell XPS 13
Embedded Linux	Skype
RTOS	HPE ProLiant
Android	Terraform

4.4. Benchmark Dataset Development

Many existing studies rely on the NVD for training and testing machine learning models. While this approach is convenient, it presents a major drawback: it does not accurately reflect real-world conditions. As a result, models trained solely on NVD data may overfit, achieving high accuracy on similar datasets but lacking robustness when applied to diverse or unseen data.

To overcome this limitation, models should be evaluated on datasets that differ in structure, content, and linguistic style, providing a more realistic measure of their generalizability. To support this need, we developed a comprehensive benchmark dataset incorporating vulnerability descriptions from CNNVD, CNVD, VarIoT, and a manually curated collection. These sources introduce greater diversity, ensuring that models are tested under conditions that closely resemble real-world challenges.

For dataset construction, we extracted vulnerability descriptions from each source and categorized them based on whether they had a CVE ID. To ensure uniqueness and prevent overlap with NVD, our benchmark dataset consists exclusively of entries without CVE IDs. This ensures that the dataset encompasses vulnerabilities not already documented in the NVD, providing a more rigorous evaluation framework.

Table 5 provides a summary of the benchmark dataset composition, detailing the number of vulnerability descriptions sourced from each database.

Table 5. Benchmark Dataset Composition

	ESV	GPSV
CNNVD	246	866
CNVD	1146	980
VarIoT	922	24
Unseen	68	100

4.5. Evaluation Metrics

To comprehensively evaluate the performance of our classification models, we employed a set of standard evaluation metrics. These metrics provide a holistic view of the models' effectiveness in classifying vulnerability descriptions.

– **Accuracy**: The proportion of correctly classified instances out of the total. It provides an overall measure of model performance.

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{Total Number of Instances}} \tag{1}$$

– **Precision**: The ratio of true positives to all predicted positives. Key for scenarios where false positives are costly.

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} \tag{2}$$

– **Recall**: The ratio of true positives to all actual positives. Important when missing a positive instance is critical.

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} \tag{3}$$

– **F1-Score**: The harmonic mean of precision and recall, useful for imbalanced datasets.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}} \tag{4}$$

– **MCC**: Matthews Correlation Coefficient (MCC) accounts for all four quadrants of the confusion matrix and provides a balanced measure, even for datasets with imbalanced classes.

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{5}$$

– **AUC-ROC**: separate predictive accuracy from threshold-specific operating conditions, offering a general measure of how well the model distinguishes between classes.

4.6. Experimental setup

A series of experiments was carried out to determine the effectiveness of our proposed ensemble embedding approach for vulnerability classification. These experiments were executed on a computer equipped with double Xion CPUs running at 2.40GHz each and 62GB of RAM, complemented by NVIDIA GeForce RTX 3090 GPUs with 24GB of VRAM. The operating system used was Ubuntu 22.04 LTS. Throughout our experiments, we made use of Tensorflow version 2.12.1 with Keras version 2.12.0.

4.7. Hyperparameter Tuning and Model Configuration

Sentence length The length of vulnerability descriptions varies significantly depending on the level of detail required to document security flaws. To identify the optimal sequence length for classification, we evaluated input lengths of 100, 200, 300, and 400 tokens, measuring both performance metrics (accuracy) and computational efficiency (training time). As shown in Figure 3, model accuracy peaks at 200 tokens (90.06%), with shorter (100 tokens: 87.57%) or longer sequences (300 tokens: 86.95%, 400 tokens: 86.33%) yielding inferior results. This decline in performance beyond 200 tokens aligns with increased noise from excessive padding or truncation, which disrupts syntactic patterns critical for accurate classification. Conversely, sequences shorter than 200 tokens lack sufficient contextual detail to resolve ambiguities in vulnerability descriptions. Training time scales linearly with sequence length, increasing from 27.33 minutes (100 tokens) to 61.11 minutes (400 tokens). While the 200-token configuration requires 38.31 minutes (41% longer than 100 tokens), its 2.5-point accuracy gain justifies this trade-off.

These results demonstrate that 200 tokens optimally balance contextual retention, classification performance, and computational efficiency. This parameter was adopted for all subsequent experiments to ensure robustness without unnecessary resource overhead.

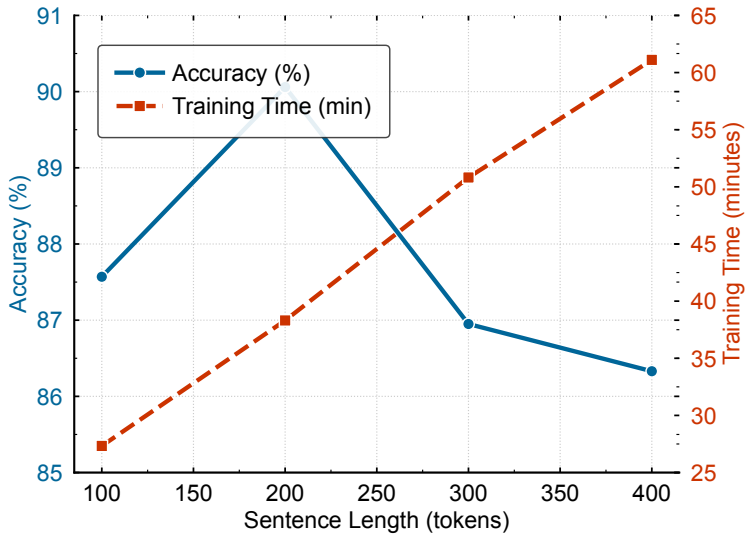

Fig. 3. Training accuracy and training time according to sentence lengths

Impact of Hidden Unit Size The dimensionality of the hidden state, or hidden size, in recurrent neural networks like BiLSTM and BiGRU is a fundamental hyperparameter that directly influences model capacity and performance. To identify the most effective configuration for our task, we systematically evaluated four distinct hidden sizes: 64, 128, 256,

and 512. The experiments, detailed in Figure 4, revealed a clear optimal point, demonstrating that model effectiveness is highly sensitive to this parameter. Our analysis indicates that a hidden size of 128 achieves the best overall performance. At this configuration, the model reaches its peak in several key metrics, including Accuracy (91.93%), F1-Score (91.50%), and Matthews Correlation Coefficient (MCC) (0.83). This result suggests that a hidden size of 128 provides an ideal balance, granting the model sufficient capacity to capture complex dependencies in the data without becoming overly complex and prone to overfitting. When the hidden size was reduced to 64, we observed a slight but consistent decrease in performance across the board. Metrics such as Accuracy (91.30%) and F1-Score (90.64%) were marginally lower than those at the optimal size of 128. This suggests that while still effective, a smaller hidden dimension may slightly limit the model's ability to learn the full complexity of the feature set, bordering on underfitting. Conversely, increasing the hidden size beyond the optimal point to 256 and 512 led to a significant degradation in model performance. For instance, the F1-Score dropped from its peak of 91.50% to 87.25% for a size of 256 and further to 87.10% for 512. A similar sharp decline was observed in the MCC, which fell from 0.83 to 0.75 and 0.74, respectively. Interestingly, while the AUC-ROC peaked at 95.12% for a size of 256, this improvement did not translate to better classification outcomes. This pattern strongly indicates that larger hidden sizes introduced excessive complexity, causing the model to overfit to the training data rather than learning generalizable patterns.

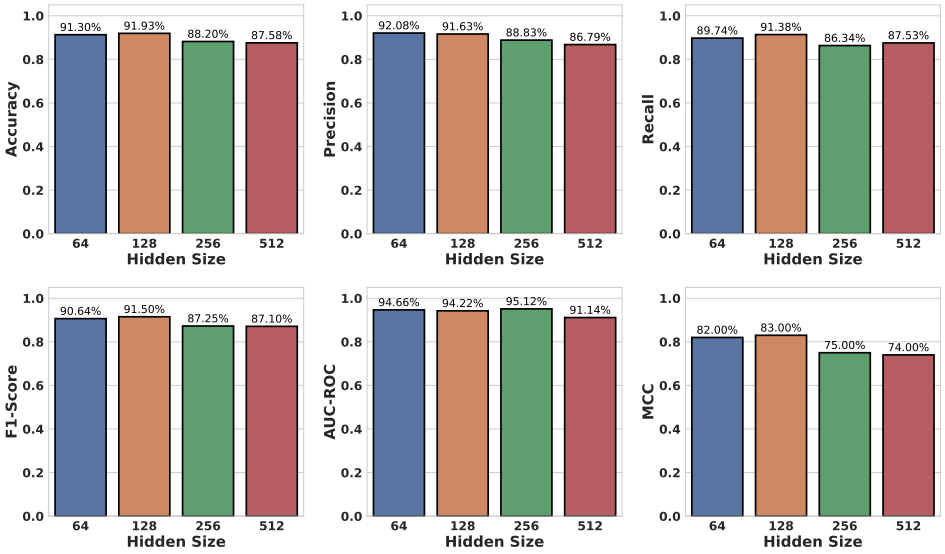

Fig. 4. Performance evaluation of ensemble embedding method across different hidden units sizes

Architectural, Optimization, and Regularization Choices Beyond the empirical tuning of sequence length and hidden unit size, several other hyperparameters were set based

on established best practices to ensure a stable, robust, and well-regularized model. Table 6 illustrates the different hyperparameters and their values. In addition to the hyperparameters in the table, batch normalization is used to avoid early overfitting of the models.

Table 6. Hyperparameters

Hyperparamters	**BiGRU**
Batch-size	32
Epochs	100
Layers	3
Droupout	0.3
Loss function	binary_crossentropy
Optimizer	Adam
Early Stop/patience	val_loss / 5
Kernel Initializer	he_uniform
Kernel Regularizer	l2(0.01)
Total params	17,816,522
Trainable params	1,411,610
Non-trainable params	16,404,912

We employed a stacked architecture of three BiGRU layers. Stacking recurrent layers allows the model to learn hierarchical representations of the input data; the initial layer captures more localized, syntactic patterns, while subsequent layers can integrate these features to learn higher-level, more abstract semantic relationships. A depth of three layers was chosen as it provides sufficient capacity for this hierarchical learning without introducing excessive computational complexity or the risk of vanishing gradients that can occur in deeper architectures.

For the optimization process, the `Adam` optimizer was selected for its widespread adoption and proven effectiveness, as it efficiently combines the advantages of momentum and adaptive learning rates. Given our binary classification task, we used `binary crossentropy` as the loss function. This is the standard choice for two-class problems, as it effectively measures the distance between the predicted probability (output by a final sigmoid activation layer) and the true binary label (0 or 1).

To prevent the model from overfitting by memorizing the training data, we implemented a suite of regularization techniques. We set a maximum of 100 `epochs`, but utilized an `Early Stopping` mechanism that monitors the validation loss and halts training if no improvement occurs for five consecutive epochs, restoring the weights from the best-performing epoch. In addition to early stopping, a `dropout` rate of 0.3 was applied after each BiGRU layer, forcing the network to learn more robust features by randomly deactivating neurons during training. To further constrain the model, `L2 regularization` (with a factor of 0.01) was applied to the kernel weights to penalize large values and promote a simpler model. Finally, to ensure stable training from the outset, the He uniform initializer was chosen. This initializer is specifically designed for layers with ReLU-like internal activations (as in GRU cells) and helps prevent vanishing or exploding gradients.

4.8. Ablation analysis

To evaluate the impact of primary embedding selection on classification performance, we conducted an ablation study comparing three ensemble architectures:

EE1: GloVe (primary) + (Word2Vec, FastText) + BiGRU
EE2: Word2Vec (primary) + (GloVe, FastText) + BiGRU
EE3: FastText (primary) + (Word2Vec, GloVe) + BiGRU

As shown in Table 7, EE3 achieves the highest performance across all metrics (Accuracy: 91.93%, F1-score: 91.63%, AUC-ROC: 94.00%), outperforming both EE1 (Accuracy: 86.96%, F1-score: 86.42%) and EE2 (Accuracy: 86.33%, F1-score: 85.18%) by significant margins. This result underscores the critical role of FastText as the primary embedding, leveraging its subword decomposition mechanism to mitigate OOV challenges. While the ensemble framework reduces OOV issues by cross-referencing secondary embeddings for missing terms using a dynamic threshold, it cannot resolve cases where a term is absent across all embeddings. FastText addresses this limitation by generating vectors for OOV words through averaged subword embeddings. However, this averaging process may dilute semantic precision for terms with non-compositional meanings.

EE2 exhibits the lowest performance, primarily due to Word2Vec's lack of subword support and case sensitivity. For instance, terms like homeplugin (lowercase) are treated as OOV in Word2Vec, even though HomePlugin (mixed case) exists in its vocabulary (Table 2). This inconsistency limits its ability to benefit from secondary embeddings, as substitutions rely on exact lexical matches. By contrast, EE1 demonstrates marginally better performance than EE2 (Accuracy: 86.96% vs. 86.33%), likely due to GloVe's global co-occurrence statistics, which capture broader semantic patterns. GloVe's robustness to syntactic variations allows it to leverage secondary embeddings more effectively, as seen in its substitution of the coordinator with the controller via FastText.

Table 7. Different architecture comparison: Test Performance Metrics

Architectures	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	AUC-ROC (%)	MCC
EE1	86.96	86.17	86.73	86.42	91.52	0.72904
EE2	86.33	86.93	84.24	85.18	90.99	0.7112
EE3	**91.93**	**91.63**	**91.38**	**91.50**	**94.00**	**0.8300**

EE1: GloVe + (Wor2Vec, FastText) + BiLGRU
EE2: Word2Vec + (GloVe, FastText) + BiLGRU
EE3: FastText + (Wor2Vec, GloVe) + BiLGRU

4.9. Comparative analysis

To measure the effectiveness of our proposed ensemble framework, we compared it against two types of models: classic word embeddings (Word2Vec, GloVe, FastText) and modern Transformer models (like BERT and RoBERTa). As shown in Table 8, we tested these models on four different datasets. Three of these, VarIoT, CNVD, and CNNVD, contain

familiar vocabulary, while the fourth, our custom Unseen dataset, was created to challenge the models with a high number of new and domain-specific terms (OOV words).

On the datasets with familiar vocabulary (VarIoT, CNVD, and CNNVD), the large Transformer models performed the best, as expected. Models like DistilBERT and ModernBERT achieved nearly perfect scores, showing the power of their deep contextual understanding. While our proposed model did not surpass these large models, it was highly competitive and consistently outperformed the classic word embeddings. For example, on the CNNVD dataset, our model's F1-score of 99.86% was a significant improvement over GloVe (97.78%) and Word2Vec (94.37%), proving it is a strong and efficient alternative.

The most important findings come from the Unseen dataset, which was designed to mimic real-world challenges with new terminology. On this dataset, the performance of the powerful Transformer models dropped significantly. BERT's F1-score, for instance, fell from 99.49% on the familiar CNVD set to just 83.25% in unseen data. This happens because they are good at adapting to the training data. However, when faced with domain-specific OOV, they fail to assign them an appropriate embedding through a weighted average of sub-tokens.

In this challenging environment, our proposed model performed better than all others, achieving the highest scores in every key metric: Accuracy (91.93%), F1-score (91.50%), and Matthews Correlation Coefficient (0.8300). This success is due to the model's flexible design. Its ability to intelligently find and substitute similar words from different embeddings using a dynamic threshold, combined with FastText's power to understand parts of unknown words via sub-word embedding, allows it to handle new terms gracefully and avoid assigning a zero vector. This result highlights our framework's key advantage: while Transformer models are excellent in predictable environments, our ensemble method is more reliable and robust for real-world software vulnerability classification tasks, where new terms appear constantly.

4.10. Limitations and Future Work

Despite the robust performance of our ensemble framework, particularly in high-OOV scenarios, it is important to acknowledge its limitations to guide future research and improvements.

- **Risk of Contextually Inappropriate Substitutions:** While our dynamic threshold is a significant improvement over a fixed value, the framework still faces a risk of contextually inappropriate substitutions. The current mechanism calculates a threshold based on the local semantic neighborhood of an OOV word within a secondary embedding. However, it does not consider the global context of the original sentence in which the word appeared. This creates a potential failure point: a secondary embedding might provide a set of seemingly good synonyms for a word in one domain (e.g., "coordinator" in networking), but if the original sentence used the word in a different domain-specific sense, the chosen substitute (e.g., "controller") might be statistically sound but semantically incorrect for that specific context.
  - **Future Work:** A promising direction to mitigate this is to incorporate a *sentence-level context validation step*. After a potential substitute is identified, its vector could be compared not just to the OOV word's vector, but to the vector representation of the entire source sentence. The substitution would only be accepted if it

Table 8. Performance Comparison of the Models on Cybersecurity Datasets.

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC	MCC
VarIoT Dataset						
Word2Vec	96.81	71.64	96.33	79.03	99.43	0.6333
GloVe	96.91	44.00	91.67	59.46	99.41	0.6232
FastText	96.60	70.64	94.19	77.68	99.33	0.6041
BERT	**99.68**	**88.89**	94.11	**91.43**	99.95	**0.9131**
ModernBERT	99.47	77.27	**100.00**	87.18	99.97	0.8767
CodeBERT	98.94	62.96	**100.00**	77.27	99.93	0.7892
DistilBERT	99.47	77.27	**100.00**	87.18	**100.00**	0.8767
RoBERTa	99.58	80.95	**100.00**	89.47	99.97	0.8978
Proposed Work	99.25	86.31	96.73	90.83	95.58	0.8238
CNVD Dataset						
Word2Vec	98.28	98.18	98.40	98.27	99.95	0.9658
GloVe	98.69	98.66	98.71	98.68	99.92	0.9737
FastText	97.67	97.63	97.68	97.65	99.68	0.9531
BERT	99.53	99.69	99.29	99.49	99.95	0.9905
ModernBERT	99.91	**100.00**	99.80	99.90	99.99	0.9981
CodeBERT	99.01	98.88	98.98	98.93	99.97	0.9801
DistilBERT	**99.95**	**100.00**	**99.90**	**99.95**	**100.00**	**0.9991**
RoBERTa	99.72	99.59	99.80	99.69	99.98	0.9943
Proposed Work	98.68	98.77	98.59	98.67	99.25	0.9737
CNNVD Dataset						
Word2Vec	96.13	97.41	92.03	94.37	99.86	0.8929
GloVe	98.41	98.84	96.81	97.78	99.97	0.9563
FastText	96.04	97.36	91.85	94.24	99.93	0.8904
BERT	99.73	99.77	99.88	99.83	99.99	0.9922
ModernBERT	99.73	99.77	99.88	99.83	**100.00**	0.9922
CodeBERT	99.28	99.20	99.88	99.54	99.97	0.9791
DistilBERT	99.64	99.54	**100.00**	99.77	**100.00**	0.9896
RoBERTa	99.37	99.20	**100.00**	99.60	99.99	0.9817
Proposed Work	**99.91**	**99.80**	99.94	**99.86**	99.99	**0.9973**
Unseen Dataset						
Word2Vec	86.95	86.55	85.88	86.18	93.22	0.7244
GloVe	88.19	87.54	87.75	87.64	92.12	0.7530
FastText	88.20	88.82	86.33	87.24	94.49	0.7512
BERT	88.85	85.29	81.31	83.25	**94.61**	0.7496
ModernBERT	86.31	82.65	75.70	79.02	92.74	0.6903
CodeBERT	85.67	82.98	72.90	77.61	92.03	0.6744
DistilBERT	85.67	77.19	82.24	79.64	92.80	0.6868
RoBERTa	88.22	83.65	81.31	82.46	93.25	0.7361
Proposed Work	**91.93**	**91.63**	**91.38**	**91.50**	**94.61**	**0.8300**

Note: Metrics (Accuracy, Precision, Recall, F1, AUC) are reported in percentages. MCC is reported as a decimal. Bold values indicate the best performance for each metric. We have also assumed that the value ‘1.00‘ for AUC-ROC was intended to be ‘100.00‘ to be consistent with the percentage-based reporting.

maintains high cosine similarity with the overall sentence context, ensuring that the replacement is not just a synonym, but the right synonym for the situation. A large pre-trained language model, such as Sentence-BERT, can be used to check sentence similarity.

- **Dependency on Subword-Capable Primary Embedding:** The success of our framework's final fallback step is highly dependent on using a subword-capable primary embedding like FastText. We have shown this is the superior architecture in our ablation study. However, this dependency makes the framework less modular. If a different, non-subword embedding model were to become state-of-the-art for a specific domain, our current ensemble method would lose its most critical safety net for handling truly novel OOV terms.
  - **Future Work:** To address this, the framework could be enhanced with an independent OOV handling module. One approach would be to train a dedicated, lightweight subword model (like fastText.zip) on the target domain corpus itself, using it exclusively as a final fallback regardless of the primary embedding. Another avenue would be to implement a trainable $¡UNK¿$ token. Instead of relying on subwords, the model could learn a specific vector representation for all unresolved OOV words during the downstream task training, allowing it to adapt to the specific types of unknown words present in the training data.
- **Computational Overhead:** The sequential nature of the fallback mechanism introduces computational overhead compared to a single embedding lookup. While this trade-off is justified by the significant accuracy gains on OOV-rich datasets, it could be a limiting factor in latency-critical applications.
  - **Future Work:** This could be optimized by creating a pre-computed OOV-substitution cache for a given domain. For a known set of domain-specific OOV words, the best substitutes could be calculated offline and stored in a dictionary for near-instantaneous lookups. For a more advanced solution, knowledge distillation could be explored, where a larger, more complex teacher model (our ensemble) is used to train a smaller, faster student model to mimic its behavior, effectively "baking in" the ensemble's knowledge into a single, efficient model.

5. Conclusion

The escalating security challenges posed by interconnected embedded systems in critical infrastructure demand innovative approaches to vulnerability management. To address this, we proposed a novel ensemble embedding technique that synergistically integrates multiple word embedding methods to mitigate the impact of OOV words, a critical limitation in existing vulnerability classification frameworks. Coupled with BiGRU networks, our model achieves robust categorization of vulnerabilities while distinguishing between embedded and general-purpose system vulnerabilities with high precision.

A key contribution of this work is the creation of a comprehensive benchmark dataset, curated from diverse vulnerability databases, which addresses the scarcity of domain-specific data for embedded systems security research. Our experiments demonstrate the framework's superior performance in identifying embedded system-specific vulnerabilities, particularly in high-OOV scenarios, outperforming the baseline word embeddings in accuracy, recall, and semantic coherence.

References

1. Aghaei, E., Shadid, W., Al-Shaer, E.: Threatzoom: Cve2cwe using hierarchical neural network. arXiv preprint arXiv:2009.11501 (2020)
2. Ben Yahya, A., El Akhal, H., El Alaoui, A.E.B.: Machine learning-based collection and analysis of embedded systems vulnerabilities. In: Enhancing Performance, Efficiency, and Security Through Complex Systems Control, pp. 242–261. IGI Global (2024)
3. Chen, J., Kudjo, P.K., Mensah, S., Brown, S.A., Akorfu, G.: An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection. Journal of Systems and Software 167, 110616 (2020)
4. Franceschi-Bicchierai, L.: How this internet of things stuffed animal can be remotely turned into a spy device, https://www.vice.com/en/article/how-this-internet-of-things-teddy-bear-can-be-remotely-turned-into-a-spy-device/, accessed: Dec 2025
5. Han, Z., Li, X., Xing, Z., Liu, H., Feng, Z.: Learning to predict severity of software vulnerability using only vulnerability description. In: 2017 IEEE International conference on software maintenance and evolution (ICSME). pp. 125–136. IEEE (2017)
6. Huang, G., Li, Y., Wang, Q., Ren, J., Cheng, Y., Zhao, X.: Automatic classification method for software vulnerability based on deep neural network. IEEE Access 7, 28291–28298 (2019)
7. Ikegwu, A.C., Uzuegbu, V., Alo, U.: Review of embedded systems and cyber threat intelligence for enhancing data security in mobile health. vol 7, 98–120 (2025)
8. Na, S., Kim, T., Kim, H.: A study on the classification of common vulnerabilities and exposures using naïve bayes. In: Advances on Broad-Band Wireless Computing, Communication and Applications: Proceedings of the 11th International Conference On Broad-Band Wireless Computing, Communication and Applications (BWCCA–2016) November 5–7, 2016, Korea. pp. 657–662. Springer (2017)
9. Newman, Lily Hay: A new pacemaker hack puts malware directly on the device (Accessed: Dec 2025), https://www.wired.com/story/pacemaker-hack-malware-black-hat/, accessed: Dec 2025
10. Nie, S., Liu, L., Du, Y.: Free-fall: Hacking tesla from wireless to can bus. Briefing, Black Hat USA 25, 1–16 (2017), https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf
11. Papp, D., Ma, Z., Buttyan, L.: Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In: 2015 13th Annual Conference on Privacy, Security and Trust (PST). pp. 145–152. ieee (2015)
12. Prigg, M.: How to get green lights all the way to work: Hackers reveal how simple it is to control traffic lights in major cities using just a laptop https://www.dailymail.co.uk/sciencetech/article-2730096/How-green-lights-way-work-Hackers-reveal-simple-control-traffic-lights-major-cities-u html, accessed: Dec 2025
13. Rios, A., Lwowski, B.: An empirical study of the downstream reliability of pre-trained word embeddings. In: Proceedings of the 28th International Conference on Computational Linguistics (COLING 2020) (2020)
14. Sharma, R., Sibal, R., Sabharwal, S.: Software vulnerability prioritization using vulnerability description. International Journal of System Assurance Engineering and Management 12, 58–64 (2021)
15. Spanos, G., Angelis, L.: A multi-target approach to estimate software vulnerability characteristics and severity scores. Journal of Systems and Software 146, 152–166 (2018)
16. Stanislav, M., Beardsley, T.: Hacking iot: A case study on baby monitor exposures and vulnerabilities. Rapid7 Report (2015)

17. Vishnu, P., Vinod, P., Yerima, S.Y.: A deep learning approach for classifying vulnerability descriptions using self attention based neural network. Journal of Network and Systems Management 30, 1–27 (2022)
18. Wang, J.A., Guo, M.: Vulnerability categorization using bayesian networks. In: Proceedings of the sixth annual workshop on cyber security and information intelligence research. pp. 1–4 (2010)
19. Wang, Q., Gao, Y., Ren, J., Zhang, B.: An automatic algorithm for software vulnerability classification based on cnn and gru. Computers & Security p. 103070 (2022)
20. Wang, Q., Gao, Y., Ren, J., Zhang, B.: An automatic classification algorithm for software vulnerability based on weighted word vector and fusion neural network. Computers & Security 126, 103070 (2023)
21. Wen, T., Zhang, Y., Wu, Q., Yang, G.: Asvc: An automatic security vulnerability categorization framework based on novel features of vulnerability data. J. Commun. 10(2), 107–116 (2015)

Aissa Ben Yahya is a Ph.D. Student in the Department of Computer Science and Applications at Moulay Ismail University, Morocco. He received his Master's degree in Networks and Embedded Systems from Moulay Ismail University, Morocco, in 2020. His research interests include using deep learning and Natural Language Processing (NLP) techniques for vulnerability classification and detection, deep learning for NLP, and the use of advanced NLP models such as transformers for various tasks.

Hicham El Akhal received his master's degree in Networks and Embedded Systems from Moulay Ismail University, Morocco, in 2020. He is currently working toward a Ph.D. degree in the Department of Information Science, Faculty of Sciences, at Moulay Ismail University, Meknes, Morocco. His research interests include artificial intelligence and agriculture.

Abdelbaki El Belrhiti El Alaoui is working as a Professor in the Department of Information Science, Faculty of Sciences Meknes, Moulay Ismail University, Morocco. He received his the Ph.D. degree form Metz University, France, in 2002, and the accreditation to supervise research degree from Moulay Ismail University, in 2009. His research areas include Wireless Sensor Networks, Network Security, Embedded Systems and Internet of Things.

Received: March 14, 2025; Accepted: November 29, 2025.