VIET: A Tool for Extracting Essential Information from Vulnerability Descriptions for CVSS Evaluation

Siqi Zhang[1], Mengyuan Zhang[1], and Lianying Zhao[2]

[1] The Hong Kong Polytechnic University, HKSAR, China
[2] Carleton University, Canada
siqi0510.zhang@connect.polyu.hk,mengyuan.zhang@polyu.edu.hk
lianying.zhao@carleton.ca

Abstract. Security vulnerabilities can be patched in order based on their severity as indicated by an assigned score, to minimize the chance and impact of potential exploits. However, it often takes several days for an analyst to assign a CVSS score, leaving a window of opportunity for attackers. Existing solutions heavily rely on the accuracy of current vulnerability scoring, which suffers from potential bias and errors introduced by human analysts. In this paper, we propose *VIET*, a tool that extracts essential information from vulnerability descriptions, which can be loosely mapped to CVSS metrics, and facilitates vulnerability evaluation. We trained a dedicated cybersecurity linguistic model based on 209,842 vulnerability descriptions and implemented a Bidirectional LSTM network trained on 800 labeled vulnerability descriptions. We evaluate the effectiveness of *VIET* through F1-scores and the efficiency of the models in terms of training time. The results show that *VIET* can extract essential entities to determining a vulnerability's severity level, reducing analysis time and addressing inconsistencies in the evaluation process.

Keywords: Cybersecurity · NER · Vulnerability Assessment.

1 Introduction

Security vulnerabilities often lead to serious consequences for individuals, organizations, and society as a whole. Unpatched vulnerabilities can be exploited by cyber-criminals to launch attacks, such as ransomware, phishing, and denial-of-service. Due to such attacks, the number of compromised personal data has reached 300 million pieces in the beginning of the year 2023 [1]. The Common Vulnerabilities and Exposures (CVE) program [2] and the National Vulnerability Database (NVD) [3] are commonly used by organizations to discover and patch vulnerabilities before they are exploited by attackers. The CVE program assigns a unique CVE-ID to each of the vulnerabilities submitted by the public, which is then received by NVD for analysis and assigning a Common Vulnerability Scoring System (CVSS) [4] score. There are 199,786 vulnerabilities (March 2023) in the database, which are fetched by hundreds of security products [5] and used in security migration/network hardening solutions [6].

However, based on data collected for vulnerabilities published in 2022, the analysis of a vulnerability typically takes days (median processing time is 7 days). This may result in delays in identifying and thus patching high-severity vulnerabilities. Note that as vulnerability patching incurs costs [7] (e.g., service outage, risk of introducing new issues, manpower), it usually needs to be prioritized based on severity in enterprise environments. Furthermore, the process of assigning CVSS scores could be influenced by potential bias and limitations of individual analysts. To make things worse, erroneous or inconsistent evaluation may lead to a wrong decision in vulnerability prioritization systems that rely on CVSS scores for the severity level.

Existing solutions mainly focus on utilizing deep learning approaches to process vulnerability description text, e.g., one-layer shallow Convolutional Neural Network (CNN) [8], and multi-task learning approach [9], and predict a multi-class severity level of vulnerabilities. These methods heavily depend on the accuracy of *the current vulnerability scoring*, which is used as input for training. The aforementioned bias and inconsistency can be detrimental to the effectiveness and reliability, e.g., the same description text in different vulnerability entries may be assigned very different severity levels.

Instead of relying on existing scoring, this paper first reviews a large corpus of vulnerability descriptions and identifies the essential information needed for severity evaluation. Based on this, we propose a tool for extracting entities in vulnerability information, called *VIET*, which can facilitate the severity evaluation of vulnerabilities. It can reduce analysis time and help address inconsistencies in the evaluation process (see Section 2.2 and the example in Section 3). Specifically, we follow a novel data science pipeline to implement such a tool. First, we trained a dedicated cybersecurity linguistic model based on 9.7M words from 209,842 vulnerability descriptions (we include the rejected vulnerability descriptions to enlarge the possible corpus); Second, we implemented a Bidirectional long short-term memory (BiLSTM) network trained based on 40K labeled words from 800 vulnerability descriptions. Third, we conducted experiments to evaluate the effectiveness of our tool. Therefore, our main contributions are threefold:

– To the best of our knowledge, in the context of utilizing vulnerability descriptions, this is the first effort towards a novel entity extraction method for identifying vulnerability entities for assessing the severity level of vulnerabilities. The labeled data for training, to the best of our knowledge, is the largest dataset based on vulnerability descriptions.
– We train a specialized linguistic model for cybersecurity. This model is based on a large corpus of 9.7 million words from 209,842 vulnerability descriptions, which includes all the existing vulnerability descriptions. By training this model, this paper advances the state-of-the-art in cybersecurity NLP and creates a valuable resource for future research.
– We evaluate the effectiveness and efficiency of *VIET* based on different lengths of the vulnerability descriptions from different years. Our results demonstrate the effectiveness and efficiency of our solution.

Table 1: CVSS V3

Base Score							
Attack Vector	Attack Comp.	Privileges Req.	User Interaction	Conf.	Inte.	Ava.	Scope
Network	Low	None	None	None	None	None	Unchanged
Adjacent							
Local	high	Low	Required	Low	Low	Low	Changed
Physical		High		High	High	High	

The remainder of the paper is organized as follows. Section 2 provides background information and a motivating example. Section 3 details the *VIET* methodology. Section 4 discusses the datasets and the technique used to process the datasets. Section 5 presents the experiment results. Section 6 reviews the related work and Section 7 concludes the paper with future directions.

2 Preliminaries

In this section, we first explain a few terms to facilitate subsequent discussions. Then, we present a motivating example and discuss the technical challenges of our work.

2.1 NVD and CVSS Metrics

NVD. The National Vulnerability Database (NVD)[3] is a government-maintained database of vulnerability information that follows industry standards. The NVD receives updates from MITRE's Common Vulnerability and Exposures (CVE) List, ensuring the database is up to date. Each CVE record in the CVE List is supplemented with additional information on the NVD website, such as severity scores, weakness enumeration, and security checklist references. The website also includes information on vulnerable product versions, components, attack vectors, and impact. Our paper primarily concentrates on vulnerability descriptions and CVSS vulnerability characteristics. As of march 2023, NVD contains 199,786 vulnerabilities.

CVSS. Each CVE record in NVD includes two severity scores, namely, CVSS Version 3 and CVSS Version 2, along with their respective characteristics. CVSS V2 has six basic vulnerability characteristics, namely, attack vector (AV), attack complexity (AC), authentication (AU), confidentiality (C), integrity (I), and availability (A), while CVSS V3 does not have AU but includes three new metrics: physical (P), privileges required (PR), and user interaction (UI) in vectors (See Table 1, highlighted cells are the newly introduced metrics). The new metrics in CVSS V3 aim to provide a more accurate and complete representation of the security risk of vulnerabilities. The scoring system ranges from 0 to 10, with higher scores indicating more severe vulnerabilities.

[3] `https://nvd.nist.gov`

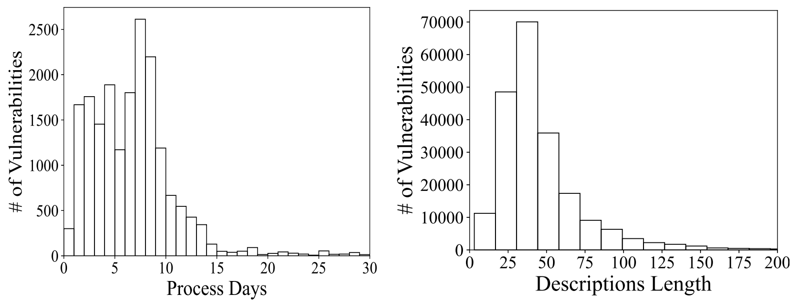


Fig. 1: Processing time (left), Length of the Descriptions (right)

Vulnerability description. CVE Numbering Authorities (CNAs) are responsible for assigning unique CVE IDs to vulnerabilities. Before assigning a CVE ID, CNAs validate the submitted vulnerability to ensure it is not already assigned or fully patched. The vulnerability description [10], which includes information such as the affected product, version, component, vulnerability type, and conditions/requirements to exploit the vulnerability, is normally provided by the reporter or created using the CVE Assignment Team's template by CNAs. The MITRE CNA of Last Resort (CNA-LR) analyzes publicly available third-party reports on vulnerabilities, known as "references". It extracts the relevant information from each reference, resolves any conflicting information or inconsistent terminology usage, and then writes the vulnerability description. Although the National Vulnerability Database (NVD) is synchronized with the CVE List, the CVE description can only be submitted through the CVE List, and the NVD has no control over the CVE descriptions. An NVD analyst may manually search through the Internet to gather any other relevant publicly available materials and information during the analysis process. Due to the large number of vulnerabilities, it may take several days before the CVSS is attributed. Figure 1 (left) shows the process time for vulnerabilities, and Figure 1 (right) shows that the most common length for a vulnerability description is around 25$\sim$70 characters.

2.2 Motivating Example

Figure 2 depicts three vulnerabilities (CVE-2017-5807, CVE-2017-5808, CVE-2017-5809) having identical vulnerability description, and further assigned to three analysts to evaluate the severity level. The three analysts may make the following decisions based on their own judgement:

– The first analyst considers this vulnerability to be easily exploitable with only remote access, as indicated by the word `remote` in the description, and also believes `arbitrary code execution` will lead to a total compromise of confidentiality, integrity, and availability (see detailed CVSS metrics in Section 2). As a result, a severity score of `10/10` is assigned to this vulnerability, which means top priority to be patched.

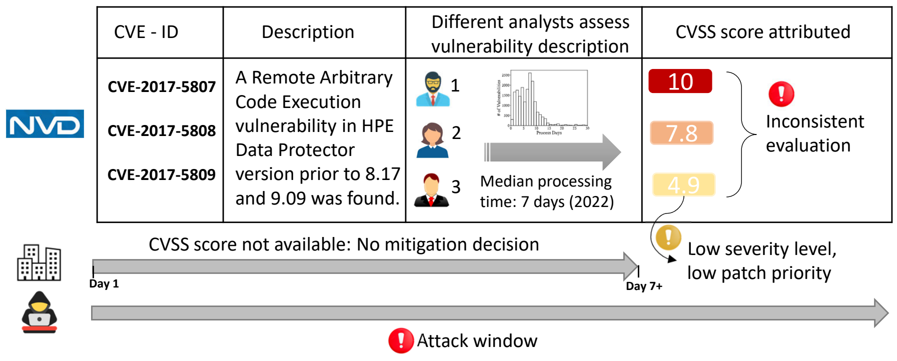
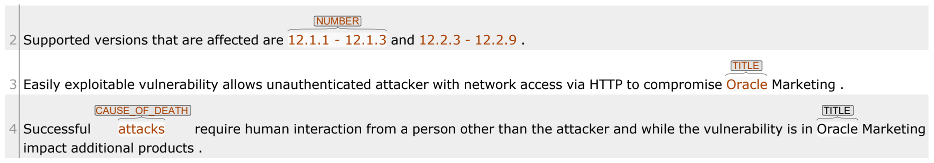

CVE - ID
Description
Different analysts assess vulnerability description
CVSS score attributed
NVD
CVE-2017-5807
CVE-2017-5808
CVE-2017-5809
A Remote Arbitrary Code Execution vulnerability in HPE Data Protector version prior to 8.17 and 9.09 was found.
1
2
3
Median processing time: 7 days (2022)
10
7.8
4.9
Inconsistent evaluation
CVSS score not available: No mitigation decision
Day 1
Day 7+
Low severity level, low patch priority
Attack window

Fig. 2: The motivating example

- The second analyst also acknowledges `remote` access as with the first analyst. Nonetheless, different impact metrics are considered, i.e., only availability is undermined if this vulnerability is exploited. Hence, `7.8/10` is assigned.
- The third analyst mistakenly considers the required access to be `local` (which is inconsistent with *"a remote arbitrary code execution"*) and only leading to information disclosure (confidentiality) as the impact of this vulnerability. Thus, this vulnerability is scored `4.9/10`.

Adding to the manual analysis time already leaving an attack window, improperly/wrongly assigned CVSS scores may further delay patching extending the attack window favoring the attacker. We thus propose to leverage NER techniques to extract vulnerability-specific entities from vulnerability descriptions for score derivation, which can serve as a new basis to complement the analysts.


2 Supported versions that are affected are 12.1.1 - 12.1.3 and 12.2.3 - 12.2.9 .

3 Easily exploitable vulnerability allows unauthenticated attacker with network access via HTTP to compromise Oracle Marketing .

4 Successful attacks require human interaction from a person other than the attacker and while the vulnerability is in Oracle Marketing impact additional products .

Fig. 3: Name Entity Recognition with CoreNLP

Challenges. To extract entities, we first evaluated vulnerability descriptions using CoreNLP [11], which is currently the state-of-the-art name entity extraction implementation. However, we found that CoreNLP failed to process vulnerability descriptions, as it often misclassifies security-related terms, such as categorizing "attacks" under the concept of "cause of death" (see Figure 3). This might be due to lacking vulnerability terminologies in the underlying linguistic model in CoreNLP. Therefore, a fine-tuned linguistic model is necessary to capture vulnerability-related entities. Previous research on attack entity extraction has focused on extracting entities related to creating attack graphs [12] and detecting inconsistencies in vulnerable versions [13]. A clear definition of the entities that are correlated with vulnerability evaluation is missing.

3 Design of *VIET*

This section first provides an overview of *VIET*, followed by the detailed methodologies for its major components.

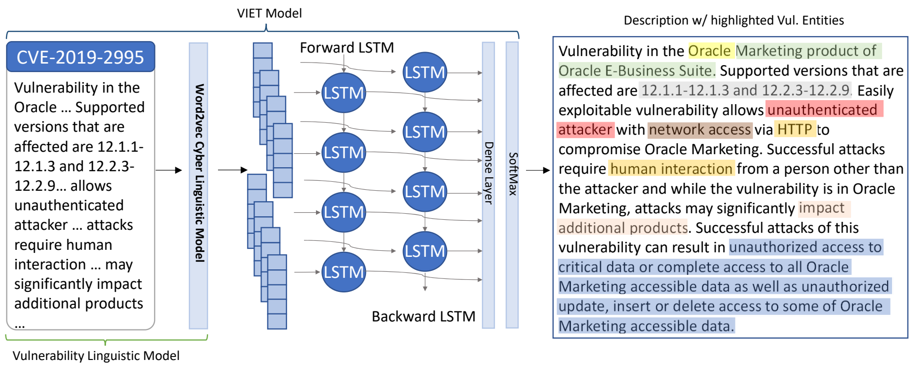

VIET Model
CVE-2019-2995
Vulnerability in the Oracle … Supported versions that are affected are 12.1.1-12.1.3 and 12.2.3-12.2.9… allows unauthenticated attacker … attacks require human interaction … may significantly impact additional products …
Word2vec Cyber Linguistic Model
Forward LSTM
LSTM
Backward LSTM
Dense Layer
SoftMax
Vulnerability Linguistic Model
Description w/ highlighted Vul. Entities
Vulnerability in the Oracle Marketing product of Oracle E-Business Suite. Supported versions that are affected are 12.1.1-12.1.3 and 12.2.3-12.2.9. Easily exploitable vulnerability allows unauthenticated attacker with network access via HTTP to compromise Oracle Marketing. Successful attacks require human interaction from a person other than the attacker and while the vulnerability is in Oracle Marketing, attacks may significantly impact additional products. Successful attacks of this vulnerability can result in unauthorized access to critical data or complete access to all Oracle Marketing accessible data as well as unauthorized update, insert or delete access to some of Oracle Marketing accessible data.

Fig. 4: The Architecture

3.1 Overview

Figure 4 depicts the two major components of *VIET*, vulnerability linguistic model and *VIET* model.

The vulnerability linguistic model is designed to train vulnerability-specific word embeddings using a dataset of 209,842 vulnerability descriptions (including rejected CVEs) from the NVD. Unlike existing works that rely on pre-trained models, our approach involves fine-tuning the linguistic model with vulnerability terminologies. Specifically, we fine-tune the linguistic model using the Word2Vec algorithm. We did not choose other language models like BERT [14] because, aside from training such complex and deep models being too expensive (while Word2Vec has only one hidden layer), specifically Binyamini et al. [12] showed that the pre-trained and fine-tuned BERT performed significantly worse than Word2Vec in such entity extraction tasks, and its visualization of the word embedding space is also poorly clustered. As a result, our approach can achieve good embedding representations with limited training data (9.7M corpus).

The *VIET* model utilizes the trained vulnerability linguistic model as input to its embedding layer. The embedding layer then feeds into a BiLSTM layer, which contains both forward and backward LSTM layers. The output from these layers is then passed through a dense layer and a Softmax layer to output the extracted vulnerability entities.

3.2 The Vulnerability Linguistic Model

In this section, we present the details of the vulnerability linguistic model with examples. We implemented both continuous bag-of-words (CBOW) and skip-gram (SG) variants of the Word2Vec algorithm. Both variants generate word embeddings that contain information about the surrounding words. In our case, we use vulnerability descriptions as inputs to fine-tune the word embeddings. In CBOW, we predict the center word from the context while in SG algorithm we try to predict the context words from the center word. In terms of training efficiency, CBOW is far better than SG, and it becomes more obvious as the corpus or window size increases, because CBOW only needs to do one representation (embedding) learning for each central word whereas SG needs (window size - 1) times. In dealing with low-frequency words, SG is more sensitive than CBOW, if there are many uncommon words in the text, SG will predict the usage environment of uncommon words to achieve a better prediction result.

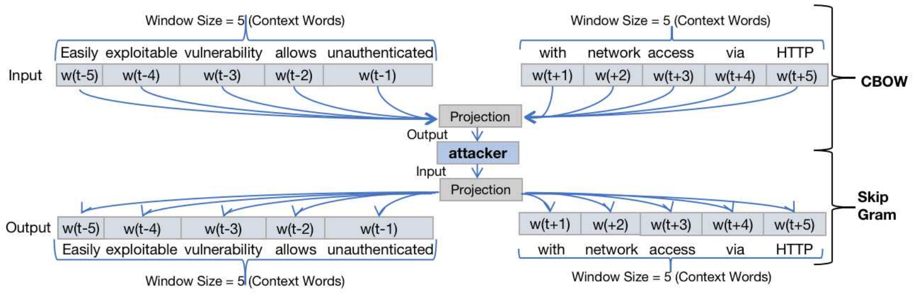

Window Size = 5 (Context Words)
Easily exploitable vulnerability allows unauthenticated
Input
w(t-5) w(t-4) w(t-3) w(t-2) w(t-1)
Window Size = 5 (Context Words)
with network access via HTTP
w(t+1) w(+2) w(t+3) w(t+4) w(t+5)
CBOW
Projection
Output
attacker
Input
Projection
Skip Gram
Output
w(t-5) w(t-4) w(t-3) w(t-2) w(t-1)
Easily exploitable vulnerability allows unauthenticated
Window Size = 5 (Context Words)
w(t+1) w(+2) w(t+3) w(t+4) w(t+5)
with network access via HTTP
Window Size = 5 (Context Words)

Fig. 5: CBOW and Skip Gram Algorithms

Example 1. Figure 5 illustrates the difference between the CBOW and SG algorithms when applied to the context "Easily exploitable vulnerability allows unauthenticated attacker with network access via HTTP". With window size 5, SG uses the word "attacker" as input to predict the surrounding 10 words "Easily, exploitable, vulnerability, allows, unauthenticated, with, network, access, via, HTTP", while CBOW takes the context set ⟨Easily, exploitable, vulnerability, allows, unauthenticated, with, network, access, via, HTTP⟩ as input and predicts the output word "attacker". In both algorithms, as a result of the training, the word "attacker" becomes closer to the set of context words ⟨Easily, exploitable, vulnerability, allows, unauthenticated, with, network, access, via, HTTP⟩ since they appear closely and frequently in the same part of the sentence. Hence, the word "attacker" will start to contain "vulnerability" meanings and be more effectively used in the vulnerability linguistic model for accurate entity extraction.

$$P(w_t \in C_t | w_c) = \frac{exp(w_c^T w_t)}{\sum_{wi \in C_t} exp(w_c^T w_i)} \quad (1)$$

Equation 1 is the softmax function to predict a probability of 1 if the $w_t \in C_t$ or 0 otherwise, where C_t is the set of all context words based on a center word, and w_c, w_t, and w_i are the embeddings of the center word, target word, and any word in all available context words, respectively.

3.3 Vulnerability Entity Extraction

Named Entity Extraction (NER) is a technique used in many areas of Natural Language Processing (NLP). The main task of this technique is to extract named entities from text, which can include things like names of people, places, organizations, and dates, among others. NER is an important tool for many NLP applications, including sentiment analysis, document classification, and question answering, among others. By identifying and extracting named entities from text, NER can help improve the accuracy and effectiveness of these applications. To identify and define the entity categories, we conducted a thorough analysis of

Table 2: The List of Vulnerability Entities (S: Supplementary, E: Essential)

	Vul. Entity	Descriptions	Example	Dist.
S	Application	Vulnerable software	Openssl, Websphere	0.0636
	CVE ID	A CVE identifier	Cve-2015-1924	0.0101
	File	Vulnerable file	wp-admin/options.php	0.0038
	Update	Service pack installation	SP1	0.0026
	Vendor	The vendor names	Microsoft, Oracle	0.0125
	Version	The application versions	12.1.0.5	0.08
	Function	The vulnerable function	getdevices()	0.0027
	Network Protocol	The network protocol	HTTP, dns	0.0026
	OS	The operating system names	Android, windows	0.0112
E	Vul. Impact	The consequences of exploiting Vul.	Obtain sensitive information	0.0766
	Vul. Type	The Vul. types	Memory corruption vulnerability	0.0574
	Vul. Complexity	The technique that used to exploit the Vul.	via crafted encrypted data	0.0322
	Vul. Vector	The context that a vulnerability gets exploited	Remote attackers, physical access	0.0317
	Privileges	The privileges that required to exploit the Vul.	Remote authenticated users	0.0169
	O	Other words do not belong any of above	a, the	0.596

the language used in vulnerability descriptions. This involved studying a large corpus of vulnerability descriptions and identifying the various types of entities that were commonly mentioned.

Our analysis revealed two main types of entities: supplementary entities and essential entities. Supplementary entities are those that cannot be directly mapped to CVSS metrics, but are still important for understanding the context of the vulnerability. Examples of supplementary entities include software names, version numbers, impacted components, and vendors, among others. On the other hand, essential entities are those that can be loosely mapped to CVSS metrics. These entities are critical for assessing the severity of a vulnerability and include things like vulnerability impact, vulnerability complexity, etc. Table 2 shows the list of vulnerability entities with definition, examples, and the distribution of each entity in the labelled data.

By defining these two entity categories, we were able to develop a more effective approach for extracting and analyzing vulnerability-specific information from textual descriptions. This allowed us to build a tool that could automatically identify and extract essential entities, making it easier for security analysts to evaluate vulnerabilities and prioritize remediation efforts.

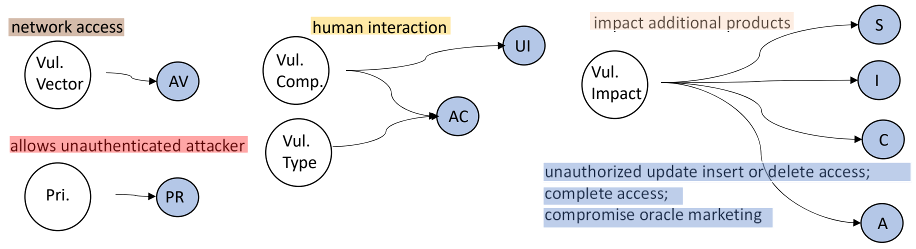

network access
Vul. Vector
AV
allows unauthenticated attacker
Pri.
PR
human interaction
Vul. Comp.
UI
AC
Vul. Type
impact additional products
Vul. Impact
S
I
C
A
unauthorized update insert or delete access;
complete access;
compromise oracle marketing

Fig. 6: A loose mapping between vulnerability entities and CVSS metrics

Example 2. Figure 6 illustrates the loose mapping between vulnerability entities and CVSS metrics. For example, with the extracted "allows unauthenticated attacker" entity from the description, the analyst can easily assign "None" to the privilege metric. Similarly, with the extracted "network access" entity, the analyst can assign "Network" to attack vector seamlessly.

4 Dataset

In this section, we first explain how to obtain our input dataset then we illustrate the techniques we used to process the dataset.

NVD entries. The NVD data feed, which provides a JSON-formatted archive of security vulnerabilities from 1999 to March 2023, contains 209K vulnerability descriptions, including those labeled as **Rejected**, with a total of 9.7M words. In order to train a vulnerability linguistic model with a larger dataset related to the cybersecurity domain, we utilized all of the vulnerability descriptions, including those with rejected CVEs.

Tokenization. We take into account multiple factors when selecting and labeling inputs, e.g., time span, as dated vulnerability scoring issues may have been fixed or not reflect new insights/versions (e.g., CVSS v3.0), reasonable quantity, as manual annotation is very time-consuming. Considering in the work of Binyamini et al. [12] 650 vulnerability descriptions were labeled (and we can target slightly higher), we used a stratified sampling method to randomly select 800 vulnerability descriptions (40K words) from the past eight years (2015-2022), with 100 descriptions from each year. Due to the presence of specific words from the cybersecurity domain, it is not ideal to use common tokenization processes like NLTK tokenization for word segmentation. Such methods may split complete links or file names into meaningless pieces due to special symbols (e.g., `admin.php?m=admin&c=site&a=save`). To preserve the meaning of cybersecurity-specific words, we used the simple `split()` function to segment the descriptions by spaces between words. In cases where there were no spaces between punctuation marks and words (e.g., "(word," "word)."), we used Regex to remove stranded whitespace and punctuation for further processing of the segmented words.

Manual labeling. To alleviate the time-consuming manual labeling process, we first employ an automatic labeling algorithm proposed by Bridge et al. [15] to label the entities such as, "application", "version", and "vendor" which reduced the proportion of "O" from 80% to 60%. However, as this autolabeling method labels vulnerability-specific entities only as relevant-terms, we have to further manually label those. Since Bridge et al. [15] used standard IOB-tagging for many multi-word names commonplace, they labeled the beginning word of an entity name with "B-X", any word in this entity name beside the beginning word is tagged with "I-X" and labeled unidentified words with "O" where X defined as a type of attack entity, we also applied this concept in the subsequent manual labeling after we generated the automatically labeled dataset through the automatic labeling process. For the subsequent manual labeling, we replaced the relevant term gazetteer that contains the words related to cybersecurity domain and 13 vulnerability categories [5] with Vulnerability type, Vulnerability impact, and Vulnerability vector which helped us to reduce the time of manual labeling. In this paper, we consider the following five essential vulnerability entities: Vulnerability vector, Vulnerability impact, Vulnerability type, Privilege, Vulnerability complexity as our mean Vulnerability vector, Vulnerability impact, Privilege, and Vulnerability complexity, 8 supplementary vulnerability entities: Vendor, Version, Application, CVE ID, Update, Function, and Network protocol, and other words not possessing any cybersecurity meaning labeled as "O". A short description and distribution of these entities are presented in Table 2.

Data imbalanced. Despite the reduction of the proportion labeled with "O" from 80% to 60%, there still exist entities that are less common such as "Network protocol" and "Privileges". To better balance the data, we applied the weighted cross-entropy loss function [12] to cope with this issue.

5 Experiment

In this section, we first provide details about how the experiment was set up and then present the results for both models.

5.1 Experiment Setup

All the experiment steps are developed in Python 3.9 and executed on a Macbook Pro running macOS Monterey, with Apple M1 Pro chip and 16.0GB of

RAM. For the vulnerability linguistic model, we leverage *Word2Vec* [16] with both algorithms, CBOW and Skip-gram, implementation from *Gensim* [17]. The embedding vector generated from the vulnerability linguistic model is fed into the embedding layer implemented based on *Keras* [18]. The deep learning model in *VIET*, e.g. BiLSTM, is implemented based on the *Keras.layers* library [18] to generate sentence embedding for the entire description and extract vulnerability entities. The evaluation metric, e.g., *loss* is implemented based on *scikit-learn* [19]. The data preprocessing and auto-labelling is based on *pandas* [20].

5.2 Evaluation of the Vulnerability Linguistic Model

To evaluate the proposed vulnerability linguistic models, we compared two fine-tuned models, CBOW and Skip-gram, with the default pretrained model. The pretrained vectors were trained on a subset of the Google News dataset, consisting of approximately 100 billion words. The model contains 300-dimensional vectors for 3 million words and phrases.[4] We chose not to use the Wikipedia pretrained model as it contains words from too many domains, and the performance of such a model is relatively slow.

The fine-tuned models take vulnerability descriptions as domain specific input and perform training based on the pretrained model. To evaluate the performance of the fine-tuned models, we examine the semantic similarities between a selected word and the top closest words. In this set of the experiment, we chose words “OS”, “priviledges”, “attackers”, and “overflow”.

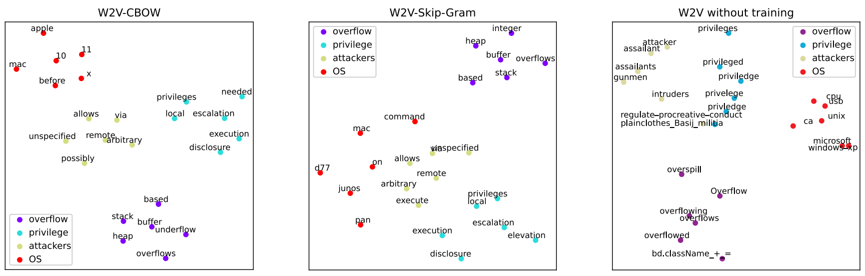
Results. The word2vec model without fine-tuning as shown in Figure 7(c) performs the worst in both separating the selected words and identifying meaningful neighbouring words. The word “attackers” cluster is very close the word “privilege” cluster. The words considered close to it are, “attacker” (the singular form of the chosen word), “assailant” (which only makes sense in natural languages), and other words that do not have a cybersecurity meaning. The word “priviledge” and “overflow” perform even worse as they only get different form or format of the original words such as, “priviledged” (the past tense), “priviledges” (the singular form), “Overflow” (the first word in upper case), “overflowing” (present participle), etc. The word “OS” cluster is surprisingly good. It contains different operating systems, e.g., “windows”, “UNIX”, and words related to “OS”, e.g., “CPU”. We assume that’s because the pretrained model is based on Google news which might have security related articles that contribute to understanding a computer science related word. However, it is evident from this experiment that the pretrained model cannot fully capture vulnerability-specific terminologies.

The word2vec-CBOW model in Figure 7(a) shows the best performance in clustering different selected words and their neighboring words. All the selected words are well separated in their respective clusters. In the “attackers” cluster, we observe the words that are used to describe attackers from vulnerability descriptions, such as, “remote”, “allows”, “arbitrary” (sample description from CVE-2015-0876: “allow remote attackers to inject arbitrary web script”), etc. The word

[4] https://huggingface.co/fse/word2vec-google-news-300

Table 3: Dataset summary and training time for the vulnerability linguistic models

Year	# of Vul. Des.	# of Corpus	W2V-CBOW	W2V-SG
1999-2023	209K	9.7M	29.67 min	124.28 min



(a) W2V CBOW (b) W2V SG (c) W2V Pretrained

Fig. 7: A visualization based on e t-Distributed Stochastic Neighbor Embedding (t-SNE) of the word embedding space for selected words based on CBOW fine-tuned model(a), skip-gram fine-tuned model(b), and the pretrained model(c)

cluster “priviledge” properly captured the words that are commonly close to it as well, e.g., “escalation” (CVE-2017-0313: “escalation of privileges”), “local” (CVE-2017-3740: “an attacker with local privileges”). This result demonstrates that with our input vulnerability description, we fine-tuned the model that captures cybersecurity-specific meanings. Similarly, the word2vec-SG model (Figure 7(b)) could also capture cybersecurity-specific meanings after fine-tuning. However, as shown in Table 3, fine-tuning the word2vec-SG model may take four times more time than the word2vec-CBOW model. The performance of both models in vulnerability entity extraction will be studied in the second experiment thoroughly.

5.3 Evaluation of the *VIET* Model

In this set of the experiments, we evaluate the performance of the *VIET* model against four different metrics, i.e., accuracy, precision, F1-score, and recall. Then we study the efficiency and scalability of the model.

Performance of vulnerability entity extraction. We train the model based on the 800 automatically and manually labeled vulnerability descriptions with different hyperparameters. The results, as shown in Table 4 and Table 5, follow the separation ratio 0.9:0.1 for the training and the testing dataset, i.e., 90% of the data goes for training and 10% for testing. We have experimented with other data separation; but the 0.9:0.1 separation yields the best performance. Although, we evaluated our model against four metrics, we choose F1-score as the

Table 4: F1-scores of the different models in essential vulnerability entity extraction: CBOW vs. SG

Models	Vul. Vector	Vul. Impact	Vul. Type	Pri.	Vul. Comp.	Macro Ave.	Weighted Ave.
CBOW	**w/o POS**						
D100-L50	0.682	0.626	0.436	0.254	0.245	0.35	0.964
D100-L100	0.752	0.682	0.555	0.444	0.458	0.453	0.972
D100-L200	0.803	0.747	0.734	0.623	0.508	0.516	0.977
D300-L50	0.764	0.704	0.701	0.357	0.342	0.492	0.972
D300-L100	0.848	0.754	0.722	0.7	0.446	0.5	0.976
D300-L200	0.851	0.773	0.746	0.757	0.505	0.58	0.977
CBOW	**w/ POS**						
D100-L50	0.878	0.795	0.809	0.706	0.625	0.5961	0.9792
D100-L100	0.916	0.807	0.846	0.723	0.679	0.604	0.980
D100-L200	0.917	0.788	0.862	0.738	0.635	0.721	0.980
D300-L50	0.901	0.793	0.835	0.752	0.679	0.655	0.981
D300-L100	0.914	0.805	0.857	0.817	0.628	0.67	0.981
D300-L200	0.902	0.807	0.86	0.831	0.719	0.68	0.9839
SG-D300-L200	0.902	0.801	0.826	0.750	0.636	0.583	0.978

main metric to compare between different models since our data is not balanced and F1-score offers the best comparison within such a dataset. We first trained the model without the part-of-speech (POS) module, then we added it back to demonstrate the improvement of such a module.

Table 5: F1-scores of the different models in supplementary vulnerability entity extraction: CBOW vs. SG

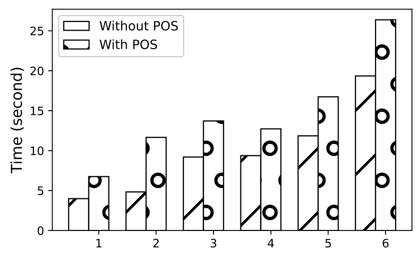
Models	Vendor	Version	Application	CVE ID	Update	Function	Network Protocol
CBOW	**w/o POS**						
D100-L50	0.522	0.637	0.367	0.35	0	0	0
D100-L100	0.641	0.716	0.565	0.836	0	0	0
D100-L200	0.675	0.746	0.617	0.852	0	0	0.286
D300-L50	0.718	0.691	0.572	0.324	0.667	0	0.4
D300-L100	0.694	0.734	0.604	0.735	0	0	0.154
D300-L200	0.675	0.7424	0.613	0.784	0.667	0	0.5
CBOW	**w/ POS**						
D100-L50	0.727	0.835	0.542	0.967	0	0.333	0.588
D100-L100	0.8	0.814	0.567	0.79	0.8	0.25	0.588
D100-L200	0.721	0.81	0.576	0.984	0.286	0.588	0.675
D300-L50	0.8	0.824	0.572	0.984	0.667	0.333	0.588
D300-L100	0.753	0.825	0.559	0.984	0.8	0.333	0.667
D300-L200	0.759	0.825	0.718	0.984	0.667	0.286	0.556
SG-D300-L200	0.741	0.452	0.636	0.984	0.000	0.333	0.588

Results. Table 4 presents the evaluation results for our model to extract essential vulnerability entities, including *vulnerability vector*, *vulnerability impact*, *vulnerability type*, *privilege*, and *vulnerability complexity*. We also provide the macro average and weighted average for each model. Our best F1-score was achieved when training the model without POS module, with 300 embedding dimensions and 200 LSTM units. The *vulnerability vector* entity achieved the highest F1-score of 0.851, followed by the *privilege* entity with an F1-score of 0.757. It is worth noting that in the closest related work [12], the F1-scores for different entities range from 0.4 to 0.94. Our results fall within this range, indicating that our approach is acceptable for this task.

To further improve the F1-score in our model, we perform training the same hyperparameter combinations with a POS module. In NLP, it's quite important to recognize parts of speech as it helps in sentence analysis and comprehension. The role a word plays in a sentence denotes its part of speech. In our design, we utilized "B-Entity" and "I-Entity" labels to identify the beginning and internal parts of entities. By integrating our labels with POS tagging, we can potentially improve the F1-score by accurately identifying entity boundaries. With 300 embedding dimensions and 200 LSTM units, our model with POS module achieves F1-score 0.902 and 0.831 for the *vulnerability vector* entity and the *privilege* entity, respectively. Despite taking four times longer to train the word2vec Skip-gram embedding model, it was only able to achieve relatively similar or slightly worse results when trained with POS tagging compared to the word2vec CBOW embedding model.

Table 5 shows the evaluation results for our model to extract supplementary vulnerability entities. Generally, the results are not as good as those for extracting essential vulnerability entities because supplementary entities are less commonly found in vulnerability descriptions. However, our results are still acceptable when we include the POS module in the training. For example, we achieve an F1-score of 0.825 for the version entity, while [12] achieves an F1-score of 0.77 for extracting versions.

Efficiency and scalability. Figure 8 shows the training time of the *VIET* model in six different hyperparameter combinations. Figure 9 illustrates the accuracy and loss versus the number of training epoch. Finally, Figure 10 demonstrates the prediction time based on different input lengths of the descriptions.



(a) Training Time

Type	**CBOW**	Training Time w/O POS	Training Time w POS

1	D = 100
	L = 50
		6.65 min	7.99 min

2	D = 100
	L = 100
		8.05 min	10.3 min

3	D = 100
	L = 300
		15.33 min	14.86 min

4	D = 300
	L = 50
		15.62 min	19.08 min

5	D = 300
	L = 100
		19.75 min	15.62 min

6	D = 300
	L = 200
		32.25 min	26.68 min

(b) Experiment D. vs. Total Training Time

Fig. 8: Training time of the Bi-LSTM model with and without POS tags vs. Different word embedding dimensions (D) and LSTM cells (L) (second per Epoch)(a), Total training time (min)(b)

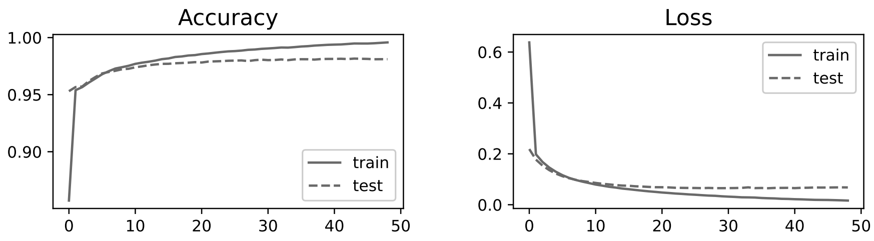


Fig. 9: Bi-LSTM model evaluation accuracy vs. epoch (left), loss vs. epoch (right)

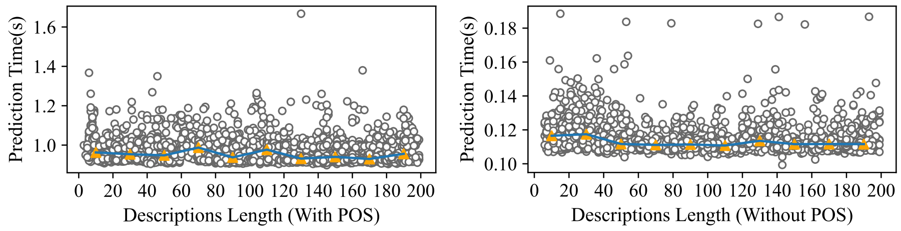


Fig. 10: Model prediction time vs. varying description lengths

Results. The training time results shown in Figure 8 indicate that the POS module introduces an additional time cost to the model training. When comparing the best performance (Type 6) in Figure 8b, the model with POS module required an extra *5s* per epoch for training. However, as early stopping was implemented in the model training, the training process stops when the validation *loss* increases while the training *loss* is still decreasing. The model with POS module only requires 26.68 minutes, while the one without POS needs 32.25 minutes to finish the training. This is because the model with POS module converges faster. The results in Figure 9 suggest that, in general, the models can reach early stop after training for only 10-20 epochs. This indicates that training the models to a reasonable level of accuracy is relatively easy and does not require a large amount of computational resources. Additionally, Figure 10 demonstrates that our solution is efficient, as the model with POS tagging takes an average of 0.96 seconds and only an average of 0.11 seconds is needed for the model trained without POS tagging to complete a prediction.

6 Related Work

In recent years, there has been a growing body of research on automating the prediction of severity scores, exploitability metrics, and impact metrics of vulnerabilities since vulnerability assessment is still primarily done manually. Researchers have focused on utilizing Natural Language Processing (NLP) to address various security problems, including vulnerability assessment.

One of the earliest works in the field of automated vulnerability assessment was conducted by Weerawardhana et al. [21]. They developed a tool that utilized

Stanford Named Entity Recognition (NER) to automatically extract software names, versions, and impacts from vulnerability descriptions from the NVD.

To predict the CVSS exploitability metrics and impact metrics by extracting information from vulnerability descriptions, the majority of research has focused on three main learning approaches: single-task, multi-target, and multi-task. Yamamoto et al. [22] were the first to use descriptions of software vulnerabilities from the NVD with a supervised Latent Dirichlet Allocation topic model to predict these CVSS metrics. Gong et al. [9] utilized multi-task learning to predict CVSS characteristics based on the vulnerability descriptions. Each character had its own label and classifier that used a shared sentence representation based on a Bi-LSTM model with an attention mechanism for capturing implicit correlations among related vulnerability characteristics. There are also six well-known ML models that most researchers used to do vulnerability classification: Naïve Bayes (NB) [23], Logistic Regression (LR) [24], Support Vector Machine (SVM) [25], Random Forest (RF) [26], XGBoost - Extreme Gradient Boosting (XGB) [27] and Light Gradient Boosting Machine (LGBM) [28]. As shown in their results, these data-driven methods performed well with satisfactory accuracy in extracting information and predicting characteristics or severity scores from vulnerability descriptions. However, predicting scores for new vulnerabilities from description text, but based on existing past CVSS scores that were assigned by security analysts still suffers from the bias/inconsistency issues we pointed out earlier, such as the same vulnerability description being assigned different severity levels. Therefore, we choose to directly perform vulnerability entity extraction from description text without relying on existing scores, which can be subsequently mapped to exploitability and impact metrics in the CVSS score system to either derive a score, or provide a reference for security analysts, improving accuracy and reducing analysis time.

7 Conclusion

We have developed a novel, end-to-end tool for extracting vulnerability entities from the description text of a security vulnerability. This tool enables security analysts to easily identify the essential vulnerability entities that loosely map to the CVSS metrics. This does not rely on the existing CVSS scores avoiding potential bias or inaccuracy introduced by human analysts. In particular, we have trained a vulnerability linguistic model on a corpus of 9.7M words from over 200K vulnerability descriptions. This linguistic model can be applied to many other NLP tasks in the vulnerability domain. Furthermore, we have manually labeled 800 vulnerability descriptions containing 40K words, which is, to the best of our knowledge, the largest dataset for vulnerability descriptions. This dataset as well as the models can be utilized to further automate vulnerability evaluation and scoring, which we consider as future work.

References

1. TechTarget, “List of Data Breaches and Cyber Attacks in February 2023 –

29.5 Million Records Breached." https://www.techtarget.com/searchsecurity/feature/Publicly-disclosed-US-ransomware-attacks-in-2023.
2. The MITRE Corporation, "CVE Website." https://cve.mitre.org/.
3. The National Institute of Standards and Technology, "NVD Data Feeds." https://nvd.nist.gov/vuln.
4. P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Security & Privacy*, vol. 4, no. 6, pp. 85–89, 2006.
5. CVEdetails, "List of Products." https://www.cvedetails.com/product-list.php.
6. L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security* (V. Atluri, ed.), vol. 5094 of *Lecture Notes in Computer Science*, pp. 283–296.
7. M. Anjum, S. Singhal, P. Kapur, S. K. Khatri, and S. Panwar, "Analysis of vulnerability fixing process in the presence of incorrect patches," *J. Syst. Softw.*, vol. 195, jan 2023.
8. H. Zhuobing, L. Xiaohong, X. Zhenchang, L. Hongtao, and F. Zhiyong, "Learning to predict severity of software vulnerability using only vulnerability description," pp. 125–136, 2017.
9. X. Gong, Z. Xing, X. Li, Z. Feng, and Z. Han, "Joint prediction of multiple vulnerability characteristics through multi-task learning," in *International Conference on Engineering of Complex Computer Systems (ICECCS'19)*, pp. 31–40, IEEE, 2019.
10. The MITRE Corporation, "How are the CVE Record DESCRIPTIONS created or compiled?." https://www.cve.org/ResourcesSupport/FAQs.
11. C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pp. 55–60.
12. H. Binyamini, R. Bitton, M. Inokuchi, T. Yagyu, Y. Elovici, and A. Shabtai, "A framework for modeling cyber attack techniques from security vulnerability descriptions," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021* (F. Zhu, B. C. Ooi, and C. Miao, eds.), pp. 2574–2583, ACM, 2021.
13. Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang, "Towards the detection of inconsistencies in public security vulnerability reports," in *28th USENIX security symposium (USENIX Security'19)*, pp. 869–885, 2019.
14. J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), pp. 4171–4186, Association for Computational Linguistics, 2019.
15. R. A. Bridges, C. L. Jones, M. D. Iannacone, K. M. Testa, and J. R. Goodall, "Automatic labeling for entity extraction in cyber security," 2014.
16. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2013.

17. R. Rehurek and P. Sojka, “Gensim–python framework for vector space modelling,” *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, vol. 3, no. 2, 2011.
18. F. Chollet *et al.*, “Keras: Deep learning for humans,” 2015. available at `https://github.com/fchollet/keras`.
19. L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.
20. W. McKinney *et al.*, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, Austin, TX, 2010.
21. S. Weerawardhana, S. Mukherjee, I. Ray, and A. Howe, “Automated extraction of vulnerability information for home computer security,” in *International Symposium on Foundations and Practice of Security*, pp. 356–366, Springer, 2014.
22. Y. Yamamoto, D. Miyamoto, and M. Nakayama, “Text-mining approach for estimating vulnerability score,” in *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pp. 67–73, IEEE, 2015.
23. S. J. Russell, *Artificial intelligence a modern approach.* Pearson Education, Inc., 2010.
24. S. H. Walker and D. B. Duncan, “Estimation of the probability of an event as a function of several independent variables,” *Biometrika*, vol. 54, no. 1-2, pp. 167–179, 1967.
25. C. Cortes and V. Vapnik, “Support-vector networks,” vol. 20, pp. 273–297, Springer, 1995.
26. T. K. Ho, “Random decision forests,” in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.
27. T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.
28. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in neural information processing systems*, vol. 30, 2017.