VulnBERTa: On automating CWE weakness assignment and improving the quality of cybersecurity CVE vulnerabilities through ML/NLP

Hannu Turtiainen
University of Jyväskylä / Binare Oy
Jyväskylä, Finland
turthzu@jyu.fi / hannu@binare.io

Andrei Costin
University of Jyväskylä
Jyväskylä, Finland
ancostin@jyu.fi

***Abstract*—*Vulnerability management* is a critical industry activity driven by compliance and regulations aiming to allocate best-fitted resources to address vulnerabilities efficiently. The cybersecurity community is global; thus, the produced vulnerability reports vary in quality and perspective. To tackle the discrepancies, machine learning (ML) has shown promise in automating vulnerability assignments. While some existing ML approaches have demonstrated feasibility, there is room for improvement. Additionally, gaps remain in the literature in understanding how the specific terminology used in vulnerability databases and reports influences ML interpretation.**

In this paper, we aim to close several such gaps. First, based on the RoBERTa transformer architecture, we introduce a systematic methodology to assign CWE-related information to a vulnerability description automatically. For that purpose, we develop a cybersecurity-focused model, VulnBERTa. Second, we apply our VulnBERTa model(s) to retroactively and automatically assign CWEs to unassigned National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) entries, thus improving the quality of open cybersecurity data. Our preliminary results are on par with comparable state-of-the-art while achieving higher classification granularity and scale.

***Index Terms*—CWE, CVE, ML, AI, BERT, Vulnerabilities, Vulnerability management, VulnBERTa, CWEml**

I. Introduction

Organizations today require and apply vulnerability management in their information systems and networks to identify, prioritize, and mitigate vulnerabilities endangering their systems and solutions. Vulnerability management is vital in maintaining the security of an organization's assets and data and reducing the risk of costly incidents. Vulnerability management is also a compliance requirement, as many industries and jurisdictions require vulnerability management practices. Frameworks such as ISO/IEC 27001 (part of the ISO27k series) and SOC 2 (Service Organization Control 2) provide requirements and aid in establishing effective vulnerability management practices and processes.

Known vulnerabilities are registered to various common knowledge databases as Common Vulnerabilities and Exposures (CVEs). In essence, CVEs are a numbering and tagging system for vulnerability reports. They can pinpoint metadata, affected systems, hardware, software, packages, and their vulnerability to each other using Common Platform Enumeration (CPE) relationships. They also calculate the Common Vulnerability Scoring System (CVSS) score to represent the vulnerability's overall potential harmfulness and provide a way to compare and prioritize vulnerabilities, among other things [23]. CVEs are commonly categorized by Common Weakness Enumerations (CWEs). Overall, the CWE and CVSS assignment to a CVE, which is generally optional but highly recommended, is used to aid in vulnerability handling through automated and semi-automated means.

Based on the National Vulnerability Database (NVD) data [7], the number of reported CVEs has risen dramatically over the last decade. In early 2013, there were a little over 21,600 reported vulnerabilities in the NVD; consequently, in early 2023, there were over 121,000. The number of reports differs yearly; however, the trend is upward. Therefore, researchers and industry practitioners seek automated yet high-quality means (e.g., tools and techniques) to handle the increasing volume of critical cybersecurity data.

Natural language processing (NLP) is a field of artificial intelligence (AI) that contemplates the interaction between natural language and computers. NLP models are trained in various tasks. This paper broadly focuses on text classification. NLP has gone through staggering advances over the last few years. However, we find it is fair to say that it is unclear if these advances can be applied *as-is out-of-box* to the systematic assignment of CWE, as existing NLP frameworks have not been modeled to understand vulnerability reports, which have their particular lingo and terms. In addition, preliminary work shows that the CVE data imbalance poses challenges to applying deep pre-trained transformers to study real-life cyber threat intelligence-related articles and reports [42].

By applying combinations of NLP and machine learning (ML) techniques, with this work, we aim to formulate and empirically answer the following research questions:

- **RQ1**: How to accurately, effectively and efficiently assign a CWE to a CVE?

- **RQ2**: What useful experience for future AI, such as improved large language models (LLMs), can be extracted from these experiments?
- **RQ3**: How can a functioning and value-added dataset be created out of varying-quality vulnerability data feeds (e.g., NVD, others)?

Our main contributions can be summarized as follows:

1) **Systematic Methodology.** We propose and present a new systematic methodology implemented into an automated system for assigning a CWE (or CWE tree) to a CVE, a vulnerability description, or a bug report (in bug trackers). We detail the methodological approach in Section IV.
2) **New Tiered CVE Open-Access Dataset.** We process, transform, and subsequently release the two decades of NVD CVE vulnerabilities to a new tiered dataset. We elaborate on the datasets and dataset construction in Section III.
3) **Core Challenges.** We identify and dissect a set of core challenges that must be addressed in dealing with vulnerability databases before highly accurate, fast, and efficient AI and LLM tools can considerably advance the field of (fully-)automated vulnerability management. We elaborate on these challenges in Section III.
4) **Open Source Code and Data.** We release the models for all the tiers and a Python-based application programming interface (API) at: https://URL-BLINDED-FOR-REVIEW.

II. Related works

A. Common Vulnerability, Platform, Weakness Enumerations

Characterizing security vulnerabilities is subjective and requires a certain degree of expertise [13], [27]. Utilizing existing data to create recommendations for characterizing vulnerabilities could increase the quality of disclosed vulnerabilities. For an example, Beck and Rass [13] proposed a neural network-based tool to aid security experts in their risk assessment duties using the CVSS scoring system. The gist of the tool is to alleviate the expert from the risk aggregation task. The procedure is beneficial in systems with a high number of dependencies.

Vulnerabilities in the software's third-party libraries or native code are common in the software life cycle. In order to link these vulnerabilities to the affected software automatically, CPE is vital. However, not all security bug reports (SBRs) are labeled with CPE. Wåreus and Hell [43] harnessed NLP to assign CPEs from vulnerability summaries automatically. They utilized Bidirectional Long-Short-Term Memory (BLSTM) [25]. The output from the BLSTM was put through the Conditional Random Field (CRF) [28] layer to make the final prediction. Overall, the authors' model achieved a precision above 0.85 [43].

Wu et al. [45] proposed a new approach to an SBRD dataset construction (SBRs and NON-SBRs). The authors initially labeled a small sample set of SBRs with a selected group of experts. Then, they iterated the data over a larger sample of non-labeled SBRs via several classification algorithms with a voting strategy. The authors eventually gathered over 80,000 records and improved an existing Chromium SBR dataset.

Common Vulnerabilities and Exposures defines publicly disclosed vulnerabilities for information technology systems, packages, and software. CVEs require maintenance from time to time, such as disclosing fixes for vulnerabilities. For automating CVE metadata and fix collection, Bhandari et al. [15] proposed `CVEfixes`, an automated tool for collecting CVEs and their fixes for open-source software. Their tool automatically fetches CVEs from NVD, finds related samples and fixes from websites like Github, and enriches the information with more metadata.

B. Autoencoder, Attention, and Transformers

One of the most influential works in NLP over the last few years is the transformer network by Vaswani et al. [40]. The model was initially created and tested for translation tasks. The Transformer consists of an encoder and a decoder stack, consisting of multiple layers of self-attention and feedforward neural networks. The positionally encoded sequence embeddings are fed into the encoder, and the output of the last layer of the encoder is fed into the decoder along with a set of encoder-decoder attention layers [40].

Google's Bidirectional Encoder Representations from Transformers (BERT) model was pre-trained on a vast unlabelled dataset; thus, BERT can be combined with proper output layers and finetuned to create state-of-the-art language models in any language task. The authors point out that transfer-learning [17] with supervised data is effective. They utilized BooksCorpus [47] and English Wikipedia totaling 3.3 billion words to pre-train BERT [21]. With their efforts, they reached state-of-the-art results at the time in several NLP benchmarks such as The General Language Understanding Evaluation (GLUE) [41].

Due to the success of BERT, Liu et al. [29] presented a replication study of Devlin et al. [21]'s work. The outcome was a Robustly optimized BERT approach (RoBERTa), a slight modification of the BERT [21] model with much longer training time and a larger dataset. Among several key changes, they increased the dataset size by a factor of ten.

Transformers are also the foundation of OpenAI's Generative Pre-trained Transformer third generation (GPT-3) [18], which is one of the most capable language models up to date. It is an autoregressive model with 175 billion trained parameters and a training dataset with almost a trillion words. OpenAI has released GPT-4 [33], which will accept different image inputs alongside text. OpenAI also claims that GPT-4 can score top human-level performance on several benchmarks.

C. Natural language processing in cybersecurity knowledge bases

As mentioned, the CVE feeds are only sometimes fully or qualitatively populated, and many entries are missing critical information. To find several issues, Sanguino and Uetz [36] evaluated the CPE dictionary and the CVE feeds. Some CVE entries lack CPE identified altogether, there are plenty of software products without a CPE identifier, and older CVEs do not carry over if the CPE changes, thus creating ghost CPEs

in the CVE feed. Sanguino and Uetz [36] propose a three-step method and a tool for fixing the errors and mismatched identifiers in the databases.

Aghaei et al. [12] automatically assigned a CVE summary to a CWE. Their approach was three-fold. First, they preprocessed the summary to a normalized version. Second, they extracted the features from the input via n-gram analysis and created a multi-hot representation vector. Lastly, they used hierarchical decision-making to assign a CWE to the original summary input. As CWE entries are hierarchical, the authors first use a multi-tier approach to assign a top-tier CWE to the entry and continue through the CWE hierarchy to find the most granular CWE presentation possible.

Furthermore, Wang et al. [42] finetuned a BERT model to assign CWEs from CVE summaries. Their model achieved above 0.90 accuracy during their testing with the top ten classes of CWEs. The authors compared their model to common machine learning algorithms and found that it achieves higher accuracy with less training time.

Yamamoto et al. [46] utilized common ML algorithms to reconstruct a CVSS vector from a CVE summary text. They concluded that supervised latent Dirichlet allocation was the best-performing algorithm, so they designed an algorithm that preferred more current CVE entries.

Similarly to Wang et al. [42], Shahid and Debar [37] utilized BERT in their NLP research. The authors finetuned multiple BERT classifiers to suit each part of the CVSS vector similarly to Yamamoto et al. [46]. The resulting vector computed to the CVSS severity score closely resembled the score interpreted by experts in the CVE feed database. Bulut and Hwang [19] also introduced their framework for assessing SBRs and predicting their CVSS severity score. They tested different embeddings and models, concluding that their BERT model generally performed the best.

D. Comparison to existing works

Overall our work is closest to Wang et al. [42] and Aghaei et al. [12], with the following main differences. We employ advances of RoBERTa [29] over BERT [21], synonym vector coding, and n-gram analysis.

Moreover, we use our tiered model specification to classify 160 different CWEs compared to only 11 CWEs by Wang et al. [42] We compared our accuracy results against Wang et al. [42] with two testing datasets containing the classes (CWEs) in the [42] dataset. We managed to get comparable accuracies for our tests with superior class count (i.e., the number of different CWEs able to be an outcome of the classification). We disclose our results in Section V.

Also, compared to Aghaei et al. [12], we fully detail our dataset (i.e., creation, structure, content, composition) and will release it as open data with the final publication.

III. Dataset

In our experiments, we used the official NIST NVD [6]. There are other vulnerability databases, for example [1]–[3], [9], [10]. However, we only chose NVD as it is the most common de-facto and open-access database for cybersecurity vulnerability data. Nevertheless, there are a couple of notable limitations to the dataset. First, the presence of numerous CVEs CWE classifications, often labeled as *NVD-CWE-Other* or *CWE-noinfo*. CWE information is necessary to ensure comprehensive vulnerability assessment and mitigation efforts, impeding understanding of underlying weakness patterns. Additionally, the database contains older entries initially assigned CWEs that have since been deprecated or restructured into broader CWE categories, making their assignment to specific vulnerabilities inappropriate and potentially misleading for security practitioners seeking accurate vulnerability classification. For example, during our dataset refinement, we found that in the employed NVD snapshot, at least 45,029 CVEs (i.e., 27.2%) have a missing CWE value. Thanks to the proposed systematic methodology, we could have easily assigned a large percentage of those with our automated approach. Moreover, we found that many of those CVEs have trivial CWEs (e.g., CWE-79 for Cross-site scripting) and could have been manually assigned upon CVE creation.

A. Dataset pruning

We created our own scripts to handle the data downloading and dataset pruning. Our scripts download all the vulnerabilities from NIST feeds from 2002 to the beginning of 2022. We also utilize "modified" and "recent" data feeds in our experiments from the time of data feed download in late February 2022. We did not update the dataset for the rest of the experiments within this paper in order to keep the data consistent. It is important to note that in September 2023, NVD will retire all the legacy data feeds and rely solely on their new API endpoints [8]. Thus, we encourage using third-party tools such as FastCVE [16] by Binare Oy to fetch CVE data reliably and efficiently.

As our models are used to assign CWEs to vulnerability descriptions, we extracted only the description and the assigned CWE from the CVE data entries, resulting in a "slimmed" version of the NVD CVE dataset. Our "slimmed" has the following samples removed from all of the downloaded data:

- Deprecated entries - MITRE maintains a list of deprecated CWEs [4].
- Categories - Some SBRs with assigned CWEs, especially older entries, had a *CWE category* [1] assigned as the primary CWE value of a CVE or bug report. MITRE guidance shows that these categories should not be mapped in the future.
- Unlabelled data - These are assigned CWEs such as *NVD-CWE-Other*.

IV. Methodology

We present below the main steps of our methodology.

[1]A CWE category is not a real CWE, but rather a meta-CWE that covers a set of individual and similar/related CWEs.

A. Tools and the environment

We used PyTorch-Lightning [22] framework versions 1.6.3 and 2.0.6 in conjunction with the HuggingFace Transformers [44] library versions 4.19.1 and 4.30.1 for Python3. Other libraries such as Scikit-learn [35] versions 1.0.2 and 1.2.2, NumPy [26] versions 1.21.6 and 1.24.3, Pandas [31] versions 1.3.5 and 2.0.2, and PyTorch [34] versions 1.11.0 and 2.0.1 were also utilized. We also used Tensorboard [11] versions 2.9.0 and 2.13.0 for training log analysis and Torchmetrics [20] versions 0.8.2 and 1.0.1 for training and testing logging and performance metrics gathering. We used one to four NVIDIA Tesla P100 or V100 GPUs (graphics processing units) to train our models. The training time varied greatly by the number of samples, classes, GPUs, and epochs; however, all the models took less than 24 hours to reach our epoch limit or a plateau.

B. Dataset experiments

Before our final tiered dataset with proper dataset pruning (see Section III-A), we used a subset of the whole data and flattened the structure to have all the CWEs in one model. First, we tested how the imbalanced nature of the dataset affects our results. We did a series of model training runs with varying sample limits. By setting the lower limit of samples to a higher number, we effectively remove some of the CWEs and group others with their parent or root CWEs; therefore, by setting up a higher low sample limit, we also lower the class count. We started by limiting the sample size to ten and gradually set the limit up to 2,000. The utilized model was roberta-base with a linear classifier getting the input from the last hidden state to predict a class (e.g., CWE). The length of the classifier was always the number of classes present in each test.

From the results, we noticed that balancing the dataset improves the accuracy significantly (i.e., 0.663 [low] vs. 0.752 [high]). However, MCC did not improve at the same rate (i.e., 0.568 [low] vs. 0.593 [high]). Thus, the decision was made only to prune samples with less than ten entries in the dataset as a whole, as we preferred the granularity of the outcome.

The results of the dataset experiments are in TABLE I.

TABLE I: Testing the effects of the imbalanced dataset.

Sample size	Classes	Acc	Mcc
10+	154	0.663	0.568
50+	111	0.66	0.563
100+	88	0.688	0.559
250+	56	0.696	0.56
500+	41	0.7125	0.6
750+	29	0.71483	0.5952
1000+	22	0.7377	0.6195
2000+	17	0.752	0.593

C. Hierarchical representation of CWEs

As we can witness from FIGURE 2, the data is heavily imbalanced. Many CWEs have ten or fewer reported vulnerabilities across two decades (see FIGURE 2a). Due to their low incidence, their impact on the dataset is low (see FIGURE 2b). However, MITRE's CWE methodology is tree-based; therefore, CWEs have parent and child CWEs. Thus, we created a CWE parent or children searcher method utilizing MITRE-supplied CWE feed [5]. As Aghaei et al. [12] found good results utilizing the CWE branch system, we also wanted to use a tiered system (i.e., following the CWE branch). We took the whole dataset and started to find parents for the listed CWE. First, we got the root CWEs for each list CWE. We did not include the base categories (i.e., research concepts, hardware design) as a tier as they are not assignable as a CWE for an SBR. Therefore, our tier one consists of the highest level of CWE after the categories. Eventually, we reached five tiers of CWEs from the NVD database by going backward from the assigned CWE as far as we needed to for each tier, thus keeping the classes (CWEs) for each tier unique. Therefore, lower-tier classes contain the assigned and child CWE samples. We illustrate the dataset processing in FIGURE 1.

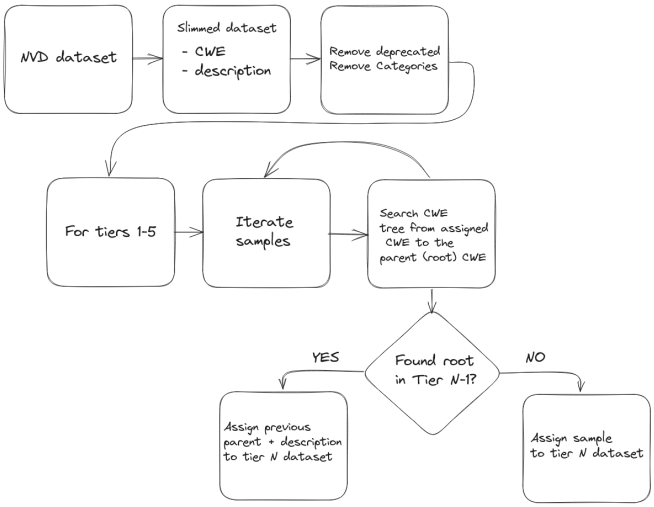


Fig. 1: Our dataset processing flow.

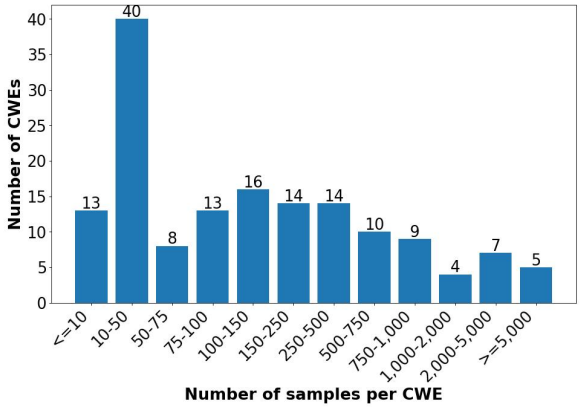
For each tier, we initially split a testing dataset from the data using Python scikit-learn [35] library's stratified dataset splitting method (we guarantee at least one sample in all the datasets). We opted to use 10% of all the data as the testing dataset. We split the remaining data into training and validation datasets, respectively, using 25% for the validation dataset. Afterward, we froze the split, saved the datasets, and used the same samples in training and testing throughout this research. TABLE II contains statistics of the tiers in our dataset. We can observe that a large portion of the entire dataset containing 165,603 samples are lost due to CWE deprecation and the removal of the category mapped and low sample count (less than ten entries in two decades of SBRs) entries. After all the dataset pruning and across all the tiers, we have a total of 160 unique CWEs, while MITRE CWE database version 4.7 recognizes 926 total unique CWEs. In comparison, without

pruning deprecated, category-assigned, or sample-size CWEs from the NVD dataset, the dataset (with previously mentioned limits) holds 291 unique CWEs.

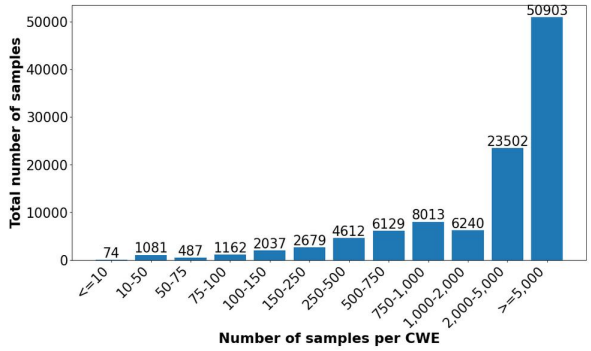
From TABLE II columns *Samples high* and *Samples low*, we can see that the dataset is highly imbalanced across the classes as we wanted to preserve a high degree of specificity by keeping the lowest required sample count as ten (10). To help alleviate the class imbalances, we calculated class weights to be considered by the model's loss function. We calculated our balancing variable as $\mu = 1.0/(total/max)$, where *total* is the total number of samples in the dataset and *max* is the maximum samples that a single class has. Then, we calculated each class's weight and smoothed them using logarithm $\omega = log(\mu * total/count)$, where *count* is the number of samples per particular class. We calculated the class weights individually for each training session and used them on every training and testing run. We also created a dictionary to match a CWE label (e.g., CWE-20) to an integer for training and testing. In NLP, the tokenizer tokenizes the text before applying it to the processing model. RoBERTa (and its predecessor BERT) have a limit of tokens per sample set to 512. We use this limit to its fullest. As the sample must always be the same size, we pad the shorter samples to 512 with zeros. Attention masks are used to tell the language model the essential parts of the sample and thus discard the padding. Our samples from NVD are mostly below that 512 tokens limit when tokenized. Out of 165,603 total samples used in all of our datasets (including unlabelled data), only 222 (0.00134%) samples are over the 512 token limit (see FIGURE 3 for reference). The tokenizer truncates those samples to 512 during training. As the number of samples over the 512 token limit is so low, we opted not to use models with larger token caps, such as the Longformer [14]. In Section IV-B, we experiment with this process's efficacy and determine other training-related measures.

D. Classifying security bug reports

We experimented with different types of classification heads on top of the RoBERTa base model. We are interested in the last hidden state output from RoBERTa for classification. The hidden state is a float tensor with a shape of (batch size, length of tokens, and output hidden size). We used a batch size of 16 for our tests and RoBERTa's default hidden size of 768. For increased training efficiency, we used batch aggregation of two. Learning rates varied between $1^{-4} - 1^{-5}$ depending on the head used. We also utilized early training, stopping when the training reached six consecutive epochs without an improvement. Other training-related configurations for all the head training experiments included cross-entropy loss as a loss function and AdamW as the optimizer with a cosine learning rate schedule with a warm-up period. We also utilized RoBERTa's built-in tokenizer. We tested our trained tokenizer for tier T1 with 50,000 token-long vocabulary using the training dataset as seen before with the dataset experiments (see Section IV-B), and it performed weaker than RoBERTa's standard. The quality and scale of the training data to train



(a) All of our data by the number of CWEs within a set sample range.



(b) All of our data by number of samples within a set sample range.

Fig. 2: Data by number of CWEs and number of samples.

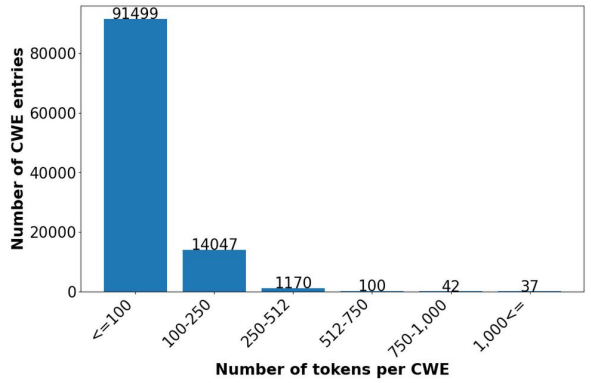


Fig. 3: Number of samples per set token range.

the RoBERTa tokenizer are, in all likelihood, vastly superior to the SBR description data; thus, the nuances and underlying structure of the text are better captured within that tokenizer.

For testing metrics, we calculated **accuracy** and **Matthews correlation coefficient** [30] for multi-class use [24]. Accuracy is defined as:

$$\text{accuracy} = \frac{\sum_{i=1}^{N} \mathbb{I}(y_i = \hat{y}_i)}{N} \quad (1)$$

where N is the total number of samples, y_i is the true label

TABLE II: Our dataset tiers.

Tier	Number of classes (CWEs)	Total number of samples	Training samples	Validation samples	Testing samples	Samples high	Samples low
T1	10	107,174	72,352	24,114	10,718	45,086	49
T2	42	106,029	71,583	23,847	10,599	30,581	12
T3	71	84,900	57,330	19,087	8,483	23,974	11
T4	30	24,720	16,695	5,556	2,469	6,895	11
T5	7	3,084	2,083	693	308	2,260	11

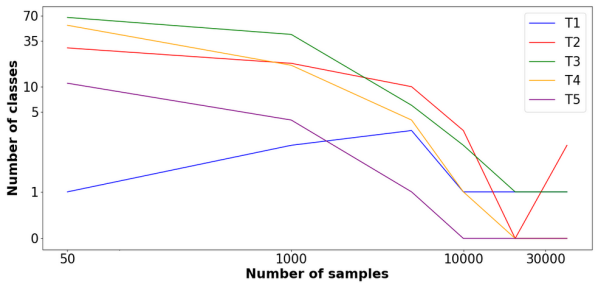


Fig. 4: Number of classes per sample range for each tier.

of the i-th sample, $\hat{y}_i$ is the predicted label of the i-th sample, and $\mathbb{I}$ is the function which returns one if the condition is true and zero otherwise. Thus, accuracy is the ratio of correctly classified samples to the total number of samples.

Matthews correlation coefficient for Torchmetrics is:

$$\text{MCC} = \frac{\sum_{k=1}^{K}\sum_{l=1}^{K} C_{k,l}C_{l,k}}{\sqrt{\sum_{k=1}^{K}(\sum_{l=1}^{K} C_{k,l})(\sum_{k=1}^{K} C_{k,l})}} \tag{2}$$

where K is the number of classes, C is the confusion matrix, where $C_{i,j}$ represents the number of samples whose true label is i and predicted label is j. In other words, the MCC measures the correlation between the true and predicted labels, considering possible imbalances in the distribution of the labels. MCC ranges from -1 to +1, where 0 indicates no correlation and -1 reverse correlation. We elaborate on the results in Section V.

V. Evaluation

In TABLE III, we have our classification head results, and in TABLE IV, we have the classifier definitions.The accuracies of the best models for each tier are promising, ranging from 0.885 to 0.977. Conversely, MCC is relatively weak for tiers T1 and T5, 0.721 and 0.596, respectively. MCC for our results indicates that although our accuracy reaches excellent levels, the accuracy per class is lacking. We believe that this is due to the imbalanced nature of the data. In FIGURE 4, we have plotted the tiered data for the number of classes per the sample ranges. The flatter the line, the more discrepancies there are between the number of samples between the classes. Tiers T2–T4 have more classes in the lower sample count region, and we have flatter lines in tiers T1 and T5. Tier T1 especially has 1-to-3 classes for all the sample ranges; thus, tier T1 is imbalanced. Tier T3, on the other hand, has many classes in the low sample count and only a few in the higher sample counts; thus, the MCC for it is higher. Noticeably, tier T2 has quite a few classes with a low to medium number of samples, no classes in between, and a few high sample count classes.

Although we tested several variations for the classification head, the results were similar overall. However, instead of supplying the head with the class token as input, we took the mean pooled dimension from all token tensors and supplied that instead. Overall, it improved our results across the board. Nevertheless, we encourage using RoBERTa's built-in classification class for future research as it performs exceptionally without requiring rigorous testing or advanced customizations.

We tested our tiered approach by appending all the testing datasets and randomly picking 10,000 samples from the combined pool (total 28,068 samples). We ran the samples through our tiered models and saved the results from each tier. With this test, our highest confidence prediction was the exact one with 2,600 samples and one of the predictions with 6,247 additional samples. Furthermore, with additional 363 samples, the parent of our best prediction was the assigned CWE. If we only note our prediction with the highest confidence, we achieved 0.26 accuracy. However, it is pretty tricky to balance the tiered models among each other; thus, we prefer always to output predictions from each tier. Suppose we calculate the accuracy based on all the predictions. In that case, we achieve 0.885 accuracy, which is arguably comparable to the state-of-the-art presented in Section II, keeping in mind that we have utilized 160 CWE classes over our five models.

During the analysis of such results, we observed that NVD had indeed updated *(only) some of such vulnerabilities* after the data gathering for our project. We may only speculate that it could indicate an ever-ongoing manual triage of many CVE entries that wastes security experts' time and focus.

For our comparison tests against Wang et al. [42], we tested two datasets. However, recreating the exact CWE dataset snapshots in time is not feasible because all CVE entries are subject to updates. Nevertheless, one testing dataset contained samples from our testing dataset (not used for training), and the other had all the samples from all the datasets. We omitted the "Others" class as explained later in Section III. The larger testing dataset contains 75,430 samples, and the smaller dataset has 8,367 samples. Wang et al. [42] disclose 20% testing dataset size; therefore, from their 66,107 sample dataset, about 13,221 should be the size of the testing dataset. However, the dataset would also contain the "Others" class samples. Thus, our testing datasets are somewhat different.

With the comparison tests, our more extensive testing dataset resulted in 49,093 of the highest confidence predictions, and additional 18,848 correct predictions were within

TABLE III: Results of the classification head testing in decimal notation (testing dataset).

Classifier (No.)*	Acc.	MCC
T1 base (1)	0.880	0.715
T1 base, mean pooling (2)	0.874	0.710
T1 base, max pooling (3)	0.883	**0.721**
T1 base, sum pooling (4)	0.846	0.672
T1 classifier 1 (5)	0.876	0.716
T1 classifier 2 (6)	0.879	0.710
T1 RoBERTa classifier (7)	0.877	0.713
T1 RoBERTa classifier, Mish [32] (8)	0.880	0.713
T1 RoBERTa classifier, $ReLU^2$ [38] (9)	0.881	0.719
T1 RoBERTa classifier, Mish [32], dropout (10)	0.876	0.717
T1 base lstm 1 (11)	**0.885**	0.719
T1 base lstm 2 (12)	0.875	0.710
T1 base lstm 3 (13)	0.880	0.715
T1 base blstm 1 (14)	0.878	0.717
T1 base freeze (15)	0.880	0.715
T1 base, tokenizer (16)	0.858	0.690
T2 base (1)	0.836	0.783
T3 base (1)	**0.885**	**0.842**
T4 base (1)	0.923	0.877
T5 base (1)	**0.977**	0.596
T2 RoBERTa classifier (7)	**0.842**	**0.789**
T3 RoBERTa classifier (7)	**0.885**	**0.842**
T4 RoBERTa classifier (7)	**0.925**	**0.879**
T5 RoBERTa classifier (7)	0.961	**0.600**
T2 base, mean pooling (2)	0.836	0.782
T3 base, mean pooling (2)	0.883	0.840
T4 base, mean pooling (2)	0.918	0.872
T5 base, mean pooling (2)	0.958	0.577
one model† (17)	0.750	0.735

Numbers in **bold** highlight the best performing model for the given tier.
† Results based on validation dataset. Not comparable results.
* Numbering ('No.') within the classifier column refers to classifier definitions and indexing from TABLE IV.

the prediction tree. Therefore, out of 75,430 samples, our prediction accuracy reached 0.901. As for the smaller dataset with 8,367 samples, we witnessed 5,405 highest confidence correct predictions and 1,962 within the prediction tree resulting in 0.880 accuracy. These results are similar to the ones by Wang et al. [42], albeit only marginally and insubstantially worse. Nevertheless, we must underline that our models contain 160 CWE classes – compared to Wang's 11 CWEs [42] our method brings approximately a 14.5x increase in classification scale. Moreover, we retain similar accuracy even with testing datasets with samples from all 160 CWE classes.

VI. Discussion and Future Work

Vulnerability databases are crucial in maintaining acceptable risk in software assets and their deployments. With the increase in reported known vulnerabilities, it is impossible to keep track of all of them manually on any software project past a certain complexity. Thus, automated tools are required. The data must be concise for the automated tools to be efficient and manageable. Crucially, databases have fundamental differences, such as the granularity of the classification between CVE and NVD databases [12]. In addition, the NVD database

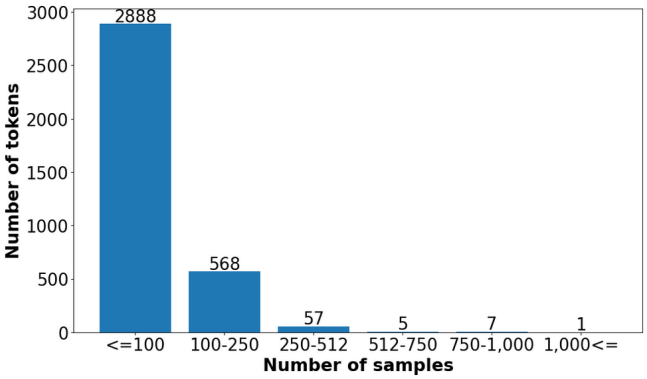


Fig. 5: Number of samples per set token range in false predictions.

has thousands upon thousands of unclassified CVE entries, of which many could be assigned quickly. Automated tools, like ours, can alleviate this discrepancy and improve the databases. Moreover, the quality can be maintained using such tools when adding new entries to the database. However, it is paramount to update such tools with new samples and possible new classes when necessary.

We see this work as the initial step and a proof-of-concept in a more elaborate and complete toolset to automatically generate a full NVD vulnerability entry from a vulnerability description or a white hat hacker walk-through of the found security issue. The aim would be an end-to-end full-concept tool for security researchers and bug hunters. In addition to CWE classification, the tool should be able to assign CVSS base score and vector and possible CAPEC entries.

To improve the CWE classification models, we will continuously (i.e., MLOps and DevSecOps) update the dataset to include newer entries and add other databases. We are also considering dropping legacy entries to focus on assigning new entries. Furthermore, we plan to also try the newer and upcoming NLP architectures to see if using those can significantly improve the results, performance, etc.

VII. Conclusion

The meticulous recording of these vulnerabilities plays a pivotal role in facilitating swift and often anticipatory countermeasures against the risks posed by these vulnerabilities. However, the efficacy of automated vulnerability management tools is contingent upon enhancing the quality of vulnerability documentation within repositories like the NVD. This enhancement is vital to fully harnessing the potential of these automated tools.

We presented a systematic methodology implemented into an automated system for assigning a CWE (or CWE tree) to a vulnerability description. Our results show near state-of-the-art performance with a large pool of possible results for maximizing the variety and granularity of the outcome. We also discussed labeling older CVE entries with unassigned CWE classifications to improve the quality and usefulness of CVE databases.

TABLE IV: Definitions of classifiers/models references in this paper and experiments.

No.	Classifier	Definitions
1	T{N} base	Tier N dataset with a single Linear layer as a head and RoBERTa class token as input
2	T{N} base, mean pooling	Tier N dataset with a single Linear layer as a head and pooled token dimensions as input
3	T{N} base, max pooling	Tier N dataset with a single Linear layer as a head and pooled token dimensions as input
4	T{N} base, sum pooling	Tier N dataset with a single Linear layer as a head and pooled token dimensions as input
5	T{N} classifier 1	Tier N dataset with classification head and pooled token dimensions as input
6	T{N} classifier 2	Tier N dataset with classification head and pooled token dimensions as input
7	T{N} RoBERTa classifier	Tier N dataset with RoBERTa classification head and pooled token dimensions as input
8	T{N} RoBERTa classifier, Mish [32]	Tier N dataset with RoBERTa-base classification head with Mish [32] activation function and pooled token dimensions as input
9	T{N} RoBERTa classifier, $ReLU^2$ [38]	Tier N dataset with RoBERTa-base classification head with $ReLU^2$ [38] activation function and pooled token dimensions as input
10	T{N} RoBERTa classifier, Mish [32], dropout	Tier N dataset with RoBERTa-base classification head with Mish [32] activation function and a larger dropout and pooled token dimensions as input
11	T{N} base lstm 1	Tier N dataset with 1-layer LSTM and mean pooling before a single Linear layer as a head
12	T{N} base lstm 2	Tier N dataset with 2-layer LSTM with dropout in between and mean pooling before a single Linear layer as a head
13	T{N} base lstm 3	Tier N dataset with 3-layer LSTM and mean pooling before a single Linear layer as a head
14	T{N} base blstm 1	Tier N dataset with 1-layer bidirectional LSTM and mean pooling before a single Linear layer as a head
15	T{N} base freeze	Tier N dataset with classification head trained with a frozen T{N} base model and RoBERTa class token as input
16	T{N} base, tokenizer	Tier N dataset with a single Linear layer as a head with self-trained tokenizer and RoBERTa class token as input
17	one model	a single model with all the applicable CWEs, architecture based on tiered base

In conclusion, we want to encourage organizations responsible for managing databases, such as NIST, to embrace the utilization of NLP technology to enhance and expand the capabilities of their already remarkable databases. The quality of data about known vulnerabilities directly correlates with the effectiveness of responses to the potential threats they outline. By leveraging NLP, we can elevate the standard of vulnerability information, leading to more robust countermeasures against the posed threats.

ACKNOWLEDGMENT

Hannu Turtiainen thanks the Finnish Cultural Foundation / Suomen Kulttuurirahasto (https://skr.fi/en) for supporting his Ph.D. dissertation work and research (grant decision no. 00231412), the Faculty of Information Technology of the University of Jyväskylä (JYU), in particular Prof. Timo Hämäläinen for partly supporting his Ph.D. supervision at JYU during 2021–2024.

(Part of) This work was supported by the European Commission under the Horizon Europe Programme, as part of the project LAZARUS (https://lazarus-he.eu/) (Grant Agreement no. 101070303). The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

(Part of this work was) Funded by the European Union (Grant Agreement Nr. 101120962, RESCALE Project). However, the views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the Health and Digital Executive Agency. Neither the European Union nor the granting authority can be held responsible.

REFERENCES

[1] Cvedetails.com the ultimate security vulnerability data source. https://www.cvedetails.com/.
[2] Cve.org by mitre. https://www.cve.org.
[3] Mend vulnerability database: The largest open source vulnerability db. https://www.mend.io/vulnerability-database/.
[4] Mitre: Common weakness enumeration deprecated entries. https://cwe.mitre.org/data/definitions/604.html.
[5] Mitre: Common weakness enumeration downloads. https://cwe.mitre.org/data/downloads.html.
[6] Nist: National vulnerability database. https://nvd.nist.gov/.
[7] Nist: National vulnerability database statistics. https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&search_type=all&isCpeNameSearch=false.
[8] Nist nvd data feeds. https://nvd.nist.gov/vuln/data-feeds.
[9] Snyk vulnerability database: Open source vulnerability database. https://security.snyk.io/.
[10] Vulndb: Number one vulnerability database documenting and explaining security vulnerabilities, threats, and exploits. https://vuldb.com/.
[11] Martín Abadi et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.
[12] Ehsan Aghaei and Ehab Al-Shaer. Threatzoom: neural network for automated vulnerability mitigation. In *6th Annual Symposium on Hot Topics in the Science of Security*, 2019.
[13] Alexander Beck and Stefan Rass. Using neural networks to aid cvss risk aggregation—an empirically validated approach. *Journal of Innovation in Digital Ecosystems*, 3(2):148–154, 2016.
[14] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *preprint arXiv:2004.05150*, 2020.
[15] Guru Bhandari, Amara Naseer, and Leon Moonen. Cvefixes: automated collection of vulnerabilities and their fixes from open-source software. In *17th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2021.
[16] "Binaré Oy"'. Fastcve - fast, rich and api-based search for cve and more (cpe, cwe, capec). https://github.com/binareio/FastCVE.
[17] Stevo Bozinovski and Ante Fulgosi. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, 1976.
[18] Tom B. Brown et al. Language models are few-shot learners, 2020.
[19] Muhammed Fatih Bulut and Jinho Hwang. Nl2vul: Natural language to standard vulnerability score for cloud security posture management. In *14th International Conference on Cloud Computing*. IEEE, 2021.
[20] Nicki Skafte Detlefsen et al. Torchmetrics - measuring reproducibility in pytorch. *Journal of Open Source Software*, 2022.
[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
[22] William Falcon. The pytorch lightning team. *Pytorch lightning*, 2019.
[23] "Forum of Incident Response and Security Teams Inc.". Common Vulnerability Scoring System v3.1: Specification Document. https://www.first.org/cvss/v3.1/specification-document, 2019.
[24] J. Gorodkin. Comparing two k-category assignments by a k-category correlation coefficient. *Computational Biology and Chemistry*, 2004.
[25] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *International Joint Conference on Neural Networks*. IEEE, 2005.
[26] Charles R. Harris et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
[27] Anne Honkaranta, Tiina Leppänen, and Andrei Costin. Toward Practical Cybersecurity Mapping of STRIDE and CWE - a Multi-perspective Approach. In *29th Conference of Open Innovations Association (FRUCT)*. IEEE, 2021.
[28] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *18th International Conference on Machine Learning*, 2001.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
[30] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.
[31] Wes McKinney et al. Data structures for statistical computing in python. In *9th Python in Science Conference*, 2010.
[32] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.
[33] OpenAI. Gpt-4 technical report, 2023.
[34] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 2019.
[35] Fabian Pedregosa et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
[36] Luis Alberto Benthin Sanguino and Rafael Uetz. Software vulnerability analysis using cpe and cve. *arXiv preprint arXiv:1705.05347*, 2017.
[37] Mustafizur R Shahid and Hervé Debar. Cvss-bert: Explainable natural language processing to determine the severity of a computer security vulnerability from its description. In *20th International Conference on Machine Learning and Applications*. IEEE, 2021.
[38] David R So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668*, 2021.
[39] Hannu Turtiainen and Andrei Costin. Vulnberta: On automating cwe weakness assignment and improving the quality of cybersecurity cve vulnerabilities through ml/nlp. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, pages 618–625, 2024.
[40] Ashish Vaswani et al. Attention is all you need. *Advances in neural information processing systems*, 2017.
[41] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.
[42] Tianyi Wang, Shengzhi Qin, and Kam Pui Chow. Towards vulnerability types classification using pure self-attention: A common weakness enumeration based approach. In *24th International Conference on Computational Science and Engineering (CSE)*. IEEE, 2021.
[43] Emil Wåreus and Martin Hell. Automated cpe labeling of cve summaries with machine learning. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2020.
[44] Thomas Wolf et al. Huggingface's transformers: State-of-the-art natural language processing. *preprint arXiv:1910.03771*, 2019.
[45] Xiaoxue Wu, Wei Zheng, Xiang Chen, Fang Wang, and Dejun Mu. Cve-assisted large-scale security bug report dataset construction method. *Journal of Systems and Software*, 160:110456, 2020.
[46] Yasuhiro Yamamoto, Daisuke Miyamoto, and Masaya Nakayama. Text-mining approach for estimating vulnerability score. In *4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, 2015.
[47] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *International Conference on Computer Vision*, 2015.