Automated Identification of Libraries from Vulnerability Data: Can We Do Better?

Anonymous Author(s)

ABSTRACT

Software engineers depend heavily on software libraries and have to update their dependencies once vulnerabilities are found in them. Software Composition Analysis (SCA) helps developers identify vulnerable libraries used by an application. A key challenge is the identification of libraries related to a given reported vulnerability in the National Vulnerability Database (NVD), which may not explicitly indicate the affected libraries. Recently, researchers have tried to address the problem of identifying the libraries from an NVD report by treating it as an extreme multi-label learning (XML) problem, characterized by its large number of possible labels and severe data sparsity. As input, the NVD report is provided, and as output, a set of relevant libraries is returned.

In this work, we evaluated multiple XML techniques and performed an analysis of different models proposed for XML classification. While previous work only evaluated a traditional XML technique, FastXML, we trained four other traditional XML models (DiSMEC, Parabel, Bonsai, ExtremeText) as well as two deep learning-based models (XML-CNN and LightXML). We compared the performance in both their effectiveness and the time cost of training and using the models for predictions. We find that other than DiSMEC and XML-CNN, recent XML models outperform the FastXML model by 3%–10% in terms of F1-scores on Top-k (k=1,2,3) predictions. Furthermore, we observe significant improvements in both the training and prediction time of these XML models, with Bonsai and Parabel model achieving 627x and 589x faster training time and 12x faster prediction time from the FastXML baseline. From a deeper analysis, we discuss the implications of our experimental results and highlight limitations that future work needs to address.

CCS CONCEPTS

• **Security and privacy** → **Software and application security**; • **Computing methodologies** → **Machine learning**; *Supervised learning*; **Natural language processing**.

KEYWORDS

multi-label classification, machine learning, vulnerability report

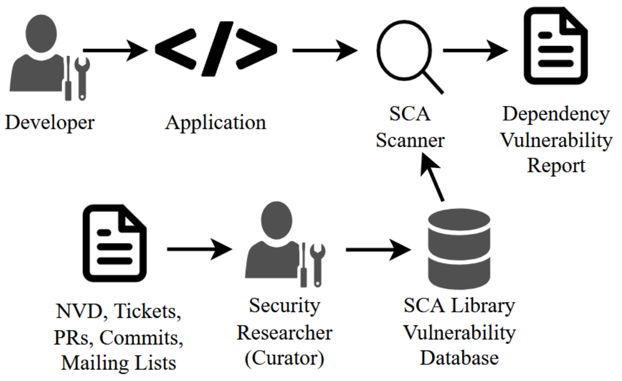

Developer
Application
SCA Scanner
Dependency Vulnerability Report
NVD, Tickets, PRs, Commits, Mailing Lists
Security Researcher (Curator)
SCA Library Vulnerability Database

Figure 1: Software Composition Analysis workflow. Security researchers have to accurately analyze vulnerability reports from NVD.

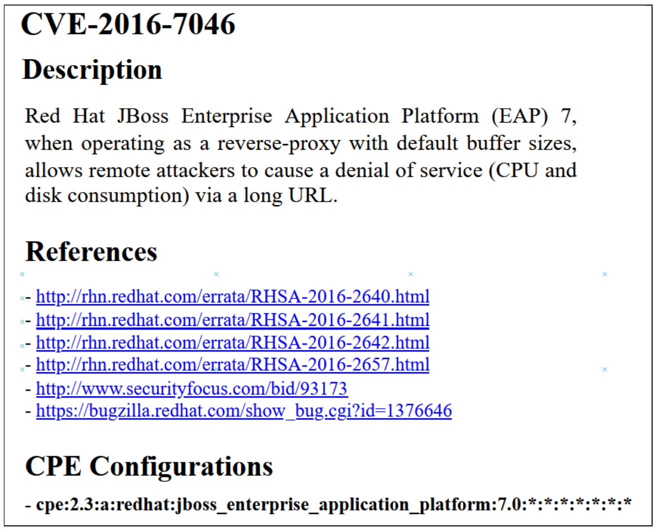
ACM Reference Format:
Anonymous Author(s). 2022. Automated Identification of Libraries from Vulnerability Data: Can We Do Better?. In *Proceedings of The 30th International Conference on Program Comprehension (ICPC 2022).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Usage of third-party libraries is crucial in software development as they help developers to build their applications by promoting software reuse. However, the use of libraries also increases the burden of software developers as they have to be aware of security vulnerabilities found in them [2, 12, 17, 28, 34, 37, 38, 47].

Software Composition Analysis (SCA) has been proposed to automatically identify vulnerable dependencies used by an application. Figure 1 shows the SCA workflow, which involves matching an application's dependencies with a database of known vulnerable libraries [11]. A team of security researchers maintains the database by monitoring and curating vulnerability data from multiple sources, including the *National Vulnerability Database* (NVD). A vulnerability report typically consists of an identification number, its CVE (Common Vulnerability Enumeration), a description of the vulnerability, references related to the vulnerability, and a set of CPE (Common Platform Enumeration) that correspond to the packages and libraries that are related to the vulnerability.

While vulnerability reports do mention related libraries, most vulnerability reports fail to include the full list of affected libraries. As an example, consider CVE-2016-7046 shown in Figure 2. In reality, the *undertow* framework, as well as the *undertow-parent* package, is affected by the vulnerability. However, its CPE and description of the vulnerability do not explicitly indicate them. After analyzing 7,666 vulnerability reports (c.f. Section 3.1), we find that 53.3% of the reports do not mention the affected libraries in



CVE-2016-7046

Description

Red Hat JBoss Enterprise Application Platform (EAP) 7, when operating as a reverse-proxy with default buffer sizes, allows remote attackers to cause a denial of service (CPU and disk consumption) via a long URL.

References

- http://rhn.redhat.com/errata/RHSA-2016-2640.html
- http://rhn.redhat.com/errata/RHSA-2016-2641.html
- http://rhn.redhat.com/errata/RHSA-2016-2642.html
- http://rhn.redhat.com/errata/RHSA-2016-2657.html
- http://www.securityfocus.com/bid/93173
- https://bugzilla.redhat.com/show_bug.cgi?id=1376646

CPE Configurations

- cpe:2.3:a:redhat:jboss_enterprise_application_platform:7.0:*:*:*:*:*:*:*

Figure 2: NVD entry for CVE-2016-7046. While the vulnerability affects the Undertow library, the term “undertow” is not explicitly mentioned in NVD.

their descriptions and CPE configurations. Human effort is needed to manually identify the affected vulnerable libraries but is slow and prone to errors. An automated approach that predicts relevant libraries from given vulnerability reports would aid the process.

A recent study by Chen et al. [11] has framed the problem as an XMTC (extreme multi-label text classification), also commonly known as XML (extreme multi-label learning), task. Our task involves the assignment of a set of relevant labels (i.e., affected libraries, manually curated by a team of security researchers) to a given document (i.e., the vulnerability report) [19]. Our task involves an extremely large number of labels (all possible libraries), and each vulnerability report may be associated with multiple labels (a vulnerability may affect multiple libraries). Indeed, our problem shares many characteristics as the XML problem, including the data sparsity problem, in which the majority of labels have only a few training instances associated with them.

Chen et al. [11] explored and evaluated the use of a traditional XML model, FastXML [40], on this task. They achieved an average F1@k score of 0.51 for k=1,2,3. Their FastXML model has been deployed within Veracode to help security researchers identify the libraries affected by vulnerability reports. Their successful experiments motivate us to apply more sophisticated models to the task. However, Chen et al. discussed the challenge of the lack of training data for the application of data-hungry techniques, such as deep-learning models. While the FastXML model considerably improves over using just the CPE configuration (average F1@k score for k=1,2,3 of 0.41) to identify libraries, we investigate and analyze the performance of multiple XML approaches, including deep-learning based approaches to better understand the applicability of XML to the problem. We also investigate the efficiency of these XML techniques to assess their feasibility for practical usage.

In this study, we conduct an investigation and experiments on the application of multiple recent XML techniques for automated library identification from vulnerability reports. Specifically, we investigate the following research question:

- **RQ1:** Do deep learning-based models and other recent XML models outperform FastXML in identifying libraries affected by a vulnerability?
- **RQ2:** How efficient are the different XML techniques?

To answer these questions, we identify six XML models that were recently proposed and have outperformed FastXML in other XML tasks. Four of them build on traditional approaches that take either a one-vs-all approach [3, 55, 56] or a tree-based approach [21, 26, 39, 40], while two models [23, 29] use deep learning. We experiment using these XML techniques in our task of automated library identification. We use the same dataset as Chen et al. [11], which contains 7,696 vulnerability reports with 4,682 labels that are collected from the NVD (National Vulnerability Database) and SCA (Software Composition Analysis) vulnerability database. To compare the effectiveness of the models, we use the same metrics as Chen et al.’s study, which are precision, recall, and F1 score at k=1,2,3. To compare the efficiency of the models, we compare the training and prediction time of all the XML techniques.

Our experiments highlight that more sophisticated XML techniques can outperform FastXML. Specifically, LightXML, a transformer-based model, achieves 10% F1 improvement in the top-k (k=1,2,3) predictions. Bonsai, a tree-based model, achieves up to 8% F1 improvement in the top-k (k=1,2,3) predictions while improving the training and prediction efficiency by 589x and 12x, respectively compared to FastXML. These results highlight that many XML techniques, including deep learning-based approaches, outperform FastXML. Despite the differences in their approach (tree-based vs. deep learning), they achieve similarly strong performance.

We also conduct a qualitative analysis of the XML models and the dataset. We compare the differences in predictions between the top-performing model and the baseline model. We also investigate why the existing XML techniques achieve strong performance despite the problem of the long tail, where many labels appear uncommonly (e.g., less than 5 times). Based on our analysis, we highlight the limitations of XML models and suggest directions for future work.

The main contributions of our work are as follows:

(1) We evaluate four traditional and two deep learning XML models on their effectiveness and efficiency on the problem of library identification from vulnerability reports. In terms of effectiveness, our results reveal that both a deep-learning based model, LightXML, and a tree-based model, Bonsai, outperform the baseline FastXML model. We find that all considered XML models are sufficiently efficient for practical usage, taking an average of less than 100 milliseconds to predict the affected libraries of a single vulnerability report.
(2) We discuss the experimental results, including a discussion of the unexpectedly strong performance of XML models on vulnerability reports with labels that do not have many training examples, known to be a problem for XML tasks.
(3) We highlight limitations of XML techniques for the library identification task, including the lack of discriminating features that can be extracted from vulnerability reports. Based on our analysis, we identify challenges for future work to address.

The rest of this paper is organized as follows. Section 2 introduces the background of our work, including XML models and existing work identifying libraries from vulnerability reports. Section 3 presents the methodology used in our study. Section 4 shows our experimental results. Section 5 discusses our findings and the lessons learnt. Finally, Section 6 concludes the paper and mentions future work.

2 BACKGROUND

2.1 Extreme Multi-Label Classification

Extreme Multi-label Learning (XML) models classify documents with relevant labels from an extremely large label space. Recent XML models can be categorized into four categories:

One-vs-all Classifiers. One-vs-all classifiers split the problem of multi label classification into multiple binary classification tasks that are independent of one another. Each binary classifier learns to distinguish a label from other labels. Classifiers in this category typically achieve good performance for XML. However, they suffer from computational and resource constraints, as they require the training of as many binary classifiers as the number of labels. One-vs-all classifier also suffers from labels that rarely occur in the dataset and has limited training data (the tail label problem). The lack of data for the tail labels leads to lower performance of the binary classifier [3]. Some one-vs-all classifiers are PDSparse [56], PPDSparse [55], and DiSMEC [3]. PDSparse and PPDSparse utilize sparse learning to reduce the complexity of one-vs-all classifiers. DiSMEC, considered the state-of-the-art for one-vs-all classifiers, uses distributed computing to reduce the complexity of learning linear classifiers for each label.

Tree-based Classifiers. Tree-based classifiers are based on decision trees, where a label tree is recursively generated based on the input features. Compared to one-vs-all classifiers, tree-based classifiers require less computational resources. However, they suffer from the tree cascading effect, where erroneous predictions in the upper nodes of the tree propagate to the lower nodes. Tree-based XML models include FastXML [40], PfastreXML [21], Parabel [39], Bonsai [26], and ExtremeText [52].

Both FastXML and PfastreXML recursively partition the feature space to build the tree, where FastXML optimizes nDCG (Normalized Discounted Cumulative Gain) based on a loss function, and PfastreXML optimizes the propensity scored loss function. Parabel and Bonsai recursively partition the label space to build a tree. Parabel partitions the labels into two balanced groups using balanced 2-mean, resulting in a deep tree. Meanwhile, Bonsai partitions the labels using K-means with a large value of K (e.g., more than 100), resulting in a wide and shallow tree. This wide and shallow tree reduces the tree cascading effect in Bonsai architecture. ExtremeText (XT) proposes an implementation of probabilistic label trees (PLT) for XML by extending hierarchical softmax (HSM) to address XML. Recent studies [26, 39] highlight that tree with label partitions (i.e., Parabel, Bonsai, and XT) outperforms tree with feature partition (i.e., FastXML and PfastreXML).

Embedding-based Classifiers. Embedding-based classifiers project the high dimensional label space into a lower dimensional space. The underlying idea for this category of approaches is that the high dimensional label space can be compressed to a lower dimensional space where similar labels have representations close to one another in the lower dimensional space. In the training process, a compressed label space is used. Then, during the prediction process, the decompressed label space is used. Thus, the label compression and decompression process are key to these classifiers. SLEEC [5] and AnnexML [46] are two proposed embedding-based classifiers for XML. SLEEC learns an ensemble of local distance preserving embeddings that preserve the pairwise distances between the nearest label vectors. AnnexML is an extension of SLEEC. It generates a K-Nearest Neighbor graph of label vectors in its embedding. The major drawback of embedding-based classifiers is the loss of information during the compression of the label space. This loss of information results in higher prediction error for embedding-based classifiers compared to other types of classifiers [26].

Deep learning-based Classifiers. Related to the embedding-based classifiers, deep learning has been utilized to learn a better representation of raw text for creating XML classifiers. Several deep learning-based classifiers for XML have been proposed [9, 23, 29, 57]. XML-CNN [29] is one of the first deep learning approaches proposed for XML problems. By using a CNN (convolutional neural network) with a hidden bottleneck layer to project the text feature into low dimensional space, XML-CNN is able to work for tasks with a large number of labels. AttentionXML [57] uses an RNN (recurrent neural network) with the attention mechanism to learn embeddings from the text inputs. Using these embeddings, AttentionXML trains a probabilistic label tree (PLT) to handle the big number of labels. Transformer-X [9] is the first deep learning XML approach to utilize transformer models, e.g. BERT [14]. Transformer-X decomposes the XML problem into a set of smaller sub-problems using label clustering. Then, the transformer model is fine-tuned to each sub-problem, creating several models in the process. However, Transformer-X still requires extensive computational resources while providing marginal improvements over AttentionXML.

More recently, to address the computational constraints of deep learning models, LightXML [23] has been proposed as a lightweight deep learning model that is trained end-to-end and has a reduced model size and training time relative to other deep learning models. LightXML fine-tunes the transformer models with dynamic negative label sampling. For its text representation, LightXML extracts embeddings from multiple layers of the transformer model. In other XML tasks, LightXML has been demonstrated to outperform other deep learning and tree-based XML classifiers.

For our study, we experiment on XML techniques that have achieved better performance in other tasks compared to FastXML, which is used in the prior work of Chen et al. [11]. From the one-vs-all classifiers, we choose DiSMEC. From the tree-based classifiers, we choose Parabel, Bonsai, and ExtremeText. From the deep learning-based classifiers, we choose XML-CNN and LightXML. These XML techniques have good performance that was evaluated and highlighted by prior works [3, 23, 26, 29, 39, 52]. The descriptions of these techniques as well as their parameters used in our experiments are provided in Section 3.4.

2.2 Existing Approaches for Library Identification from Vulnerability Report

Chen et al. [11] proposed two approaches to automatically identify affected libraries from vulnerability reports. In their study, they explored the CPE matcher and FastXML.

(1) **CPE matcher.** As a simple baseline that does not use machine learning, the CPE matcher is an approach that uses the library names listed in a vulnerability report's CPE configuration. After the library names are retrieved from the CPE configurations, they are output as the predicted affected libraries from the vulnerability report.
Overall, as the CPEs do not identify all relevant libraries, the CPE matcher achieves an average F1 of only 0.24 [11].

(2) **FastXML.** Chen et al. [11] proposed the use of FastXML. Their experiments showed that FastXML achieves an average F1-score of 0.51 at top-k predictions (k=1,2,3), outperforming the CPE matcher. A detailed description of the FastXML algorithm is provided in Section 3.4.

3 METHODOLOGY

3.1 Dataset

We utilize the dataset that was used by prior work of Chen et al. [11]. This dataset comprises 7,696 vulnerability reports with 4,682 labels (i.e., libraries) collected from the NVD (National Vulnerability Database) and the SCA (Software Composition Analysis) vulnerability database. Each report consists of a unique CVE ID, its vulnerability description, a list of web references, its CPE (Common Platform Enumeration) configuration, and its labels (i.e., the libraries that correspond to the given vulnerability report, manually curated by a team of security researchers). The information related to the vulnerability is collected from NVD entries between 2002 to 2019. For each vulnerability report, the SCA vulnerability database is used to determine the affected library names based on the vulnerability's CVE ID. Within the dataset, we find that each vulnerability is related to 1 – 432 libraries. The ground truth labels of this dataset are vetted by security experts from Veracode SCA.

3.2 Data Preparation

Before we proceed to the model training process, we perform several data preprocessing steps.

1. Data Cleaning. We check the vulnerability reports in the dataset to ensure that their information is correct and up-to-date. For this purpose, we utilize the API provided by NVD[1] and compare the information retrieved from the API with the dataset. From the 7,696 vulnerability reports that we checked, we find that 30 of them are no longer in use (e.g., no longer deemed a vulnerability, rejected by NVD, etc.). After removing these entries, our dataset has 7,666 vulnerability reports.

2. Label Merging. Next, we check the labels of the remaining 7,666 vulnerability reports and find that there are labels that always co-occur, which is a known problem in XML tasks [32, 51]. We find that the majority of co-occurrences are between labels that are closely related, such as *gnome-session* and *gnome-shell*. Therefore, we merge the labels that always co-occur into a single label. For example, the labels *gnome-session* and *gnome-shell* are merged into *gnome-session;gnome-shell*. We merged a total of 1,865 labels, reducing the number of labels from 4,682 to 2,817. Figure 3 shows the distribution of the number of labels per vulnerability.

[1]https://nvd.nist.gov/vuln/data-feeds

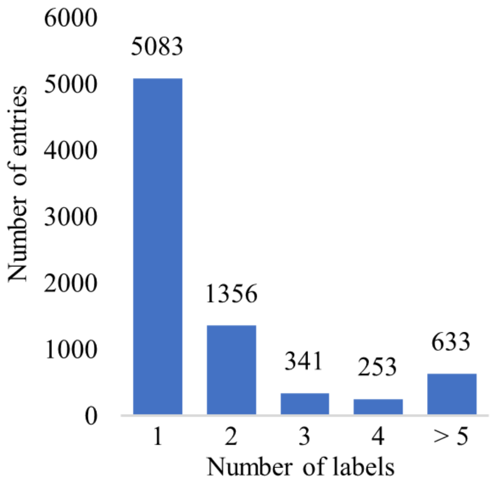


Figure 3: Distribution of the number of labels that the entries are related to

3. Feature Engineering. From each vulnerability report, we use its vulnerability description, references, and CPE configurations for training the models. We perform the same preprocessing steps as prior work [11]. While all three components are textual, we preprocess the components differently:

- *Description*: From the description text, we remove non-alphanumeric characters and non-noun words. Non-alphanumeric characters are removed using a regular expression. Meanwhile, we remove the non-noun words using the part-of-speech tagging provided by NLTK [6] library. Following Chen et al. [11] work, we also remove words that appear in more than 30% of the vulnerability data (i.e., common words). To do so, we use CountVectorizer from Scikit-Learn [36] to count the occurrences of each word in the vulnerability data.
- *References*: We remove the non-alphanumeric characters from references. As an example, consider the reference link "http://secunia.com/advisories/59328" from CVE-2014-1533. We replace non-alphanumeric characters with whitespace, resulting in the following string: `http secunia com advisories 59328`.
- *CPE configurations*: For each CPE configuration, we use a regular expression based on the CPE format [7] to retrieve the library names contained within the CPE. As an example, consider the following CPE configuration for CVE-2014-1533: `cpe:2.3:a:mozilla:firefox:*:*:*:*:*:*`. Using the regular expression, we retrieve the library name `mozilla firefox`.

After preprocessing, we combine the text of the three components into a single feature.

4. Train and Test Data Preparation. For a fair comparison between the models, we use the same training and testing dataset. Similar to Chen et al. [11], we use a 75%-25% split, where 75% of the dataset forms the training dataset while the remaining 25% forms the test dataset. As each instance in the dataset may have multiple labels, we use iterative stratification [42, 45] to split the dataset, which is recommended for experiments on multi-label datasets [10, 43]. We use the *scikit-multilearn*'s implementation[2] to split the dataset. This results in a split where the training dataset has 6,017 entries and the test dataset has 1,649 entries.

3.3 Experimental Setup

Through our experiments, we aim to evaluate the performance (to answer RQ1) and efficiency (to answer RQ2) of the XML models. In this section, we describe the device configuration and the evaluation metrics used to assess the performance and efficiency of the model.

Device Configuration. All the model training and prediction processes are done in a Docker environment running Ubuntu 18.04 with Intel(R) i7-10700K @ 3.8GHz, 64GB RAM, and 2 RTX 3070 GPU. For the XML techniques which require deep learning (XML-CNN and LightXML), we utilize the GPU for training. For the techniques that do not require deep learning (FastXML, DiSEMC, Parabel, Bonsai, ExtremeText), we utilize all available CPU cores (8 cores).

Performance Metric. Following previous work [11], we evaluate the performance of a model through precision (P), recall (R), and F1-score (F1) calculated for the top k (k=1,2,3) prediction results. In total, we have nine performance metrics: P@1, R@1, F1@1, P@2, R@2, F1@2, P@3, R@3, and F1@3. These metrics are standard metrics used for the evaluation of XML tasks in prior studies [23, 26, 39, 40, 52]. Given the top-k prediction *pred_k(vul)* and the actual labels *label(vul)* for a given vulnerability report, *precision@k* and *recall@k* are defined as follow:

$$precision@k(vul) = \frac{pred\_k(vul) \cap label(vul)}{k}$$

$$recall@k(vul) = \frac{pred\_k(vul) \cap label(vul)}{|label(vul)|}$$

Then, we compute the average of the precision and recall calculated above to obtain the *precision@k* and *recall@k* that we use to compare the performance between the XML techniques (n refers to the number of data):

$$precision@k = \frac{1}{n} \sum_{vul=1}^{n} precision@k(vul)$$

$$recall@k = \frac{1}{n} \sum_{vul=1}^{n} recall@k(vul)$$

Finally, we calculate the *F1-score@k* by using the harmonic mean of *precision@k* and *recall@k*.

$$F1@k = 2 \times \frac{precision@k \times recall@k}{precision@k + recall@k}$$

Efficiency Metric. To evaluate the efficiency of the XML models, we measure their execution time during training and prediction. The training time is measured from the start of the training process until the trained model is saved to a file. For prediction time, we measure the time required by the model to produce predictions for the entire test dataset containing 1,649 vulnerability reports. Based on this prediction time, we compute the average time required by the model to produce predictions of the affected libraries for one vulnerability report.

[2]http://scikit.ml/api/skmultilearn.model_selection.iterative_stratification.html

Table 1: FastXML training parameters

Parameter	Value
Number of trees	64
Parallel jobs	No. of CPUs
Max. leaf size	10
Max. labels per leaf	20
Re-split count	0
Subsampling data size	1 (no subsampling)
Sparse multiple	25
Random number seed	2016

Table 2: DiSMEC training parameters

Parameter	Value
Solver type	L2-regularized L2-loss support vector classification
Bias	1
Cost	1
Learning batch	1000

3.4 Model Implementation

Using the dataset that has been split into train and test data as specified in Section 3.2, we perform experiments using seven different XML models. For the three tree-based models (i.e., FastXML, Parabel, and Bonsai), the performance of the model may fluctuate slightly due to the randomness in the clustering. To mitigate the effect of randomness, we run the experiment ten times for each tree-based model and report the average of the performance metrics. For the one-vs-all and deep-learning based models, we construct one model each as the performances of these models do not fluctuate. For all models, we select parameters based on the parameters reported in previous works. If more than one set of parameters were reported, we pick the parameters used for experiments on datasets that are most similar to our dataset by comparing the total number of entries and labels.

FastXML. FastXML is a tree-based XML classifier, using trees to represent hierarchies over the feature space. An ensemble of trees are trained, and to build a tree, FastXML recursively partitions the parent node by optimizing the normalized Discounted Cumulative Gain (nDCG) as its ranking loss function. To perform prediction, FastXML returns the ranked list of the most frequently occurring labels in all the leaf nodes of the built trees ensemble. As the model used for baseline comparison, we replicate the FastXML model used by Chen et al. [11]. We use the model parameters that are listed in their paper, which can be seen in Table 1.

Table 3: Parabel and Bonsai model training parameters

Parameter	Parabel	Bonsai
Clustering function	Balanced 2-means	K-means
Cluster size	3	100
Num trees	3	3
Loss function	hinge	hinge
Maximum depth	20	20
Feature	TF-IDF	TF-IDF

Table 4: ExtremeText model training parameters

Parameter	Value
Learning-rate	1.0
L2 regularization	0.001
Tree arity	2
Word vector size	100
Feature	TF-IDF

DiSMEC. DiSMEC (Distributed Sparse Machines for Extreme Multi-label Classification) [3] is a one-vs-all XML classifier that uses a distributed learning mechanism for scalable one-vs-all model training. DiSMEC employs a binary one-vs-rest framework to learn the weight vector for each label. As the number of labels grows, more weight vectors has to be learned.

For better performance, DiSMEC takes a distributed approach, where labels are sent to training nodes in batches of 1,000. Within each node, a batch is trained in parallel [8]. Using this distributed approach, DiSMEC achieves comparable training and prediction time with tree-based XML classifiers (e.g. FastXML). In our experiments on DiSMEC, we use the training parameters listed in Table 2. These parameters were used for all experiments in the study proposing DiSMEC [3].

Parabel. Parabel [39] combines a tree-based approach and a one-vs-all approach. One-vs-all approaches tend to have higher prediction accuracies compared to tree-based classifiers. On the other hand, one-vs-all approaches have significantly higher training and prediction cost compared to tree-based approaches. Parabel aims to have a comparable training speed with tree-based approaches while maintaining a similar accuracy to the one-vs-all XML approach.

Parabel learns up to three label trees by recursively partitioning the labels into two balanced groups using *balanced 2-means clustering*. Each leaf node in a label tree is associated with a set of linear one-vs-all classifiers, one for each label contained in the leaf. Each non-leaf node within the tree is associated with binary classifiers that decide whether the currently processed data should be passed down to the left, right, or both child nodes. A piece of data may arrive into multiple leaf nodes, where the one-vs-all classifiers predict the corresponding labels for the given data. In our experiments, we use the parameters shown in the second column of Table 3. These parameters were used for all experiments in the study proposing Parabel [39].

Bonsai. Bonsai [26] works similarly to Parabel, learning label trees by partitioning the labels and having one-vs-all classifiers in its leaf node. Different from Parabel's deep and balanced tree, Bonsai creates a diverse and shallow tree. This is done by setting the K value of its K-means clustering to >2 (default to K=100 in the Bonsai implementation) and dropping the balanced trees constraint. The intuition behind the use of a shallow tree is to minimize the error propagation due to the tree cascading effect. In our experiments on Bonsai, we use the parameters shown in the third column of Table 3. These parameters were also used in all experiments in the study proposing Bonsai [26].

Table 5: XML-CNN model training parameters

Parameter	Value
Dynamic max pooling	[128, 128, 128]
Filter channel	128
Filter sizes	[2, 2, 2]
Hidden dimensions	1024
Learning rate	0.003
Stride	[2, 1, 1]
Feature	TF-IDF

Table 6: LightXML training parameters

Parameter	Value
Learning rate	0.00001
Epoch	30
Batch size	4
SWA warmup	10
SWA step	200

ExtremeText. ExtremeText (XT) [52] is a tree-based model that is built on top of FastText [24], a text classification approach using hierarchical softmax (HSM). XT extends FastText for XML problems by utilizing probabilistic label trees (PLT) [22], which generalizes HSM for multi-label classification. In building the PLT, XT uses hierarchical clustering with recursive balanced k-means until the size of the clusters are smaller than a given value (e.g., 100). This clustering approach allows for similar labels to be located close to one another within the tree. Each node in the tree is associated with a logistic regression classifier. In our experiments, we use the training parameters shown in Table 4, which are based on the parameters used for the experiments on the EURLex-4K [33] dataset in the study proposing XT [52].

XML-CNN. XML-CNN [29] is a deep learning approach for XML. XML-CNN extends the CNN (Convolutional Neural Network) proposed by Kim et al. [27]. To adapt the CNN architecture for XML tasks, XML-CNN makes some modifications in its architecture. XML-CNN uses binary cross-entropy (BCE) loss function rather than the sigmoid function, as they find that BCE loss is more suitable for XML. XML-CNN adds a hidden bottleneck layer, which is a fully-connected hidden layer between the pooling and output layer.

XML-CNN requires a validation dataset for training. Thus, to train an XML-CNN model, we further split the training dataset into 80% training data (4850 entries) and 20% validation data (1167 entries). The parameters that we use for the XML-CNN model are listed in Table 5. These parameters were used in the experiments in the study proposing XML-CNN [29].

Table 7: Experiment result on the performance of various XML modeling techniques. The two models with the best performance are highlighted in bold text. The average F1 is the arithmetic mean of F1@1, F1@2, and F1@3. Improve vs. FastXML column shows the average F1-score improvement of the model compared to FastXML. The category column refers to the category of the model based on Section 2.1.

Category	Model	P@1	R@1	F1@1	P@2	R@2	F1@2	P@3	R@3	F1@3	Avg. F1	Improve vs. FastXML
One-vs-all	DiSMEC	0.79	0.58	0.67	0.57	0.72	0.64	0.44	0.76	0.55	0.62	-3%
Deep learning	XML-CNN	0.80	0.59	0.68	0.58	0.75	0.65	0.44	0.79	0.56	0.63	-1%
Tree-based	FastXML	0.81	0.59	0.69	0.59	0.74	0.65	0.45	0.79	0.57	0.64	0%
Tree-based	ExtremeText	0.84	0.63	0.72	0.59	0.77	0.67	0.45	0.82	0.58	0.66	3%
Tree-based	Parabel	0.87	0.65	0.74	0.62	0.80	0.70	0.47	0.85	0.60	0.68	7%
Tree-based	**Bonsai**	**0.87**	**0.65**	**0.74**	**0.62**	**0.80**	**0.70**	**0.47**	**0.86**	**0.61**	**0.68**	**7%**
Deep learning	**LightXML**	**0.88**	**0.66**	**0.75**	**0.64**	**0.82**	**0.72**	**0.49**	**0.87**	**0.63**	**0.70**	**10%**

LightXML. LightXML [23] is a recent deep learning-based XML approach that takes a transformer-based approach. LightXML is a deep learning model which fine-tunes a transformer model with dynamic negative label sampling. LightXML model consists of four components: label clustering, text representation, label recalling, and label ranking. First, the labels are clustered such that each label belongs to one label cluster. Balanced 2-means clustering is used to recursively partition the label sets to create the clusters. Then, to obtain the text representation, LightXML uses transformer models which embed raw text into a high dimensional representation. For this purpose, three pre-trained transformer models are used: BERT [14], XLNet [53], and RoBERTa [30]. To reduce the computational complexity associated with the use of transformers, only the base model of each transformer is used (12 layers and 768 hidden dimensions). This text representation is the input of the label recalling and label ranking components.

For label recalling and label ranking, LightXML uses generative cooperative networks with dynamic negative label sampling. Label recalling acts as the generator that dynamically samples positive and negative labels. The label ranking part acts as the discriminator, which can distinguish between positive and negative labels. Given a raw text as an input, LightXML first takes the raw text input to construct a representation. Using the text representation, the label recalling component scores all label clusters and returns possible labels. Finally, the label ranking component scores every label returned by the label recalling component, obtaining the top-k labels. For our LightXML model training, we use the parameters shown in Table 6. These parameters are based on the parameters used in the experiments on the EURLex-4k [33] dataset in the study proposing LightXML [23].

4 RESULTS

Based on the experiment setup that we describe in Section 3.3, we conduct our experiments on six different XML techniques (DiSMEC, Parabel, Bonsai, ExtremeText, XML-CNN, and LightXML). We compare their results with the FastXML baseline that was used in Chen et al. [11] study. The following subsections provide details of the evaluation results on both the performance (RQ1) and the efficiency (RQ2) of the XML techniques.

4.1 RQ1: Do deep learning-based models and other recently proposed XML models outperform FastXML?

The results of the performance evaluation for all XML techniques are shown in Table 7. Based on the experiment results, we found that all XML techniques other than DiSMEC and XML-CNN achieve better results than FastXML on all the evaluation metrics. The biggest performance improvement can be seen in the top-1 prediction results, where ExtremeText, Parabel, Bonsai, and LightXML achieve 5%, 8%, 8%, and 10% F1-score improvement respectively. For the top-2 prediction results, ExtremeText, Parabel, Bonsai, and LightXML achieve 3%, 7%, 7%, 9% F1-score improvement respectively. For the top-3 prediction results, ExtremeText, Parabel, Bonsai, and LightXML achieve 3%, 6%, 6%, and 10% F1-score improvement respectively. Among the tested XML techniques, LightXML achieves the highest improvement from the FastXML baseline.

While LightXML outperforms the other models in terms of effectiveness, XML-CNN does not. Furthermore, the tree-based model, Bonsai, has comparable performance to LightXML. This indicates that deep learning methods do not always outperform tree-based models. Future work should, therefore, still consider XML techniques that use approaches other than deep learning-based techniques.

Comparing the performance of the XML models with the CPE matcher described in Section 2.2, we observe that all XML models achieve better performance. Even the worst performing XML model, DiSMEC, with an average F1-score of 0.62, outperforms the CPE matcher which achieves an average F1-score of only 0.24 based on Chen et al.'s [11] evaluation. This highlights that the use of machine learning approaches significantly improves accuracy in predicting affected libraries from vulnerability reports.

Apart from DiSMEC and XML-CNN, the other XML models outperform FastXML. Both the best-performing deep learning-based approach and tree-based approach obtain improvements of 5% – 10% over FastXML. The transformer-based LightXML has the greatest F1-score improvement of 10%.

Table 8: Execution time for the training and prediction of the XML models

Model	Train (s)	Prediction (s)	Avg. Pred. (ms)
Parabel	0.47	0.63	0.38
Bonsai	0.50	0.64	0.39
DiSMEC	24.28	2.55	0.58
ExtremeText	43.64	0.67	1.55
XML-CNN	90.34	3.24	1.96
FastXML	294.75	8.01	4.86
LightXML	15,378.65	103.72	62.90

4.2 RQ2: How efficient are the different XML techniques?

It is important that the trained XML models are efficient enough for practical application. The results of the efficiency evaluation for the XML techniques are shown in Table 8. All techniques, apart from LightXML, have a shorter training time compared to the baseline FastXML. In terms of prediction time, only one model, LightXML, requires a longer time to produce its predictions than FastXML. XML-CNN achieves 3x faster training time and 2x faster prediction time than FastXML, while ExtremeText achieves 6x faster training time and 11x faster prediction time. To train LightXML, over 15,378 seconds (4.27 hours) are required, which is higher than the 294 seconds required for training FastXML. Despite the decrease in efficiency, LightXML can be trained in several hours and is, therefore, still practical to be deployed for practical use. The LightXML model can be retrained overnight whenever a change in data distribution is observed (e.g. the model's effectiveness begins to drop).

The biggest improvements over FastXML can be seen in Parabel and Bonsai, which have similar training and prediction times. It takes less than a second to train both Parabel and Bonsai. Specifically, Parabel takes 0.47 seconds while Bonsai takes 0.50 seconds. These training times are equal to 627x and 589x faster training time over FastXML respectively. For prediction, Parabel takes 0.63 seconds while Bonsai takes 0.64 seconds, 12x faster than FastXML.

We also compute the average time required to predict the libraries of one vulnerability report in the test data. All of the XML models have an average prediction time of less than one second. Parabel achieves the fastest prediction time of 0.38 milliseconds for a vulnerability report. Meanwhile, LightXML requires the longest prediction time of 62.90 milliseconds for a vulnerability report, 13x more than the prediction time of FastXML model. All considered XML models, including LightXML, are practical as they produce predictions for a single vulnerability report in a fraction of a second.

Overall, the considered XML models are all practical for use in terms of efficiency. Apart from LightXML, all models have better efficiency than FastXML in both training and prediction time. LightXML underperforms all other models, requiring training time that is two orders of magnitude greater than FastXML. In turn, FastXML is an order of magnitude less efficient than the other five models. All models require just a fraction of a second for the average prediction time of one vulnerability report.

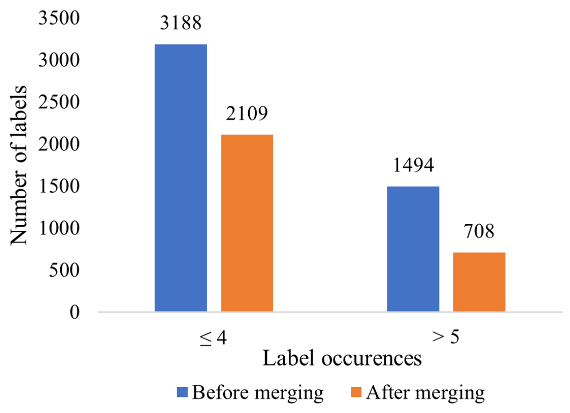


Figure 4: The number of occurrences of the labels in the dataset.

5 DISCUSSION

5.1 Lessons Learned

Transformer-based deep learning model achieves the best performance but is less efficient. Based on the experimental results described in Section 4, we find that among the tested XML techniques, LightXML is the best-performing model. Our results are consistent with other studies in text classification in multiple tasks, in which transformer-based models outperform other models [1, 35, 49, 54]. However, the improvements in the effectiveness are not without a cost. Transformer-based models require more computational resources and take more time for both training and prediction. This limitation can be seen in the training and prediction time of LightXML that is shown in the last row of Table 8. LightXML takes 15,378.65 seconds (4 hours 16 minutes, and 18.65 seconds) for training and 103.72 seconds (1 minute 43.72 seconds) for making 1,649 predictions. This is greater than the training and prediction times of other XML models. For example, Bonsai takes less than one second for both training and prediction.

Knowing this characteristic of the transformer-based model, several considerations can be made when choosing an XML model. If short training and prediction time is important, then it may be better to use a tree-based model, such as Bonsai, which achieves only slightly worse performance than LightXML but has higher efficiency. Meanwhile, if time and computational resources are not a problem, the transformer-based model achieves the best performance. In the future, another possible approach is to reduce the required resource and time of the transformer model. LightXML utilize three transformer models, namely BERT [14], RoBERTa [30], and XLNet [53]. It is possible to either reduce the number of transformer models used (e.g., only using one transformer model such as BERT) or to use lightweight transformer models that have been proposed by recent studies [16, 31, 44].

Ease of predicting the tail labels. Prior studies [26, 48] raised the problem that improvements in the effectiveness of better XML models may come only from the better prediction of the labels with a large number of training data (the head labels), while still performing poorly on the labels with a small number of training

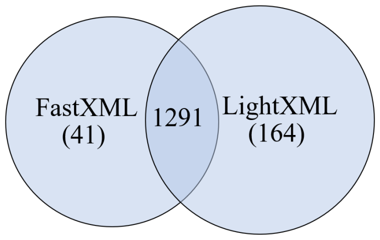


Figure 5: Top-1 prediction differences and intersections between FastXML and LightXML model.

data (the tail labels). This is undesirable in a practical setting with real-world implications, such as our task. Therefore, it is important to understand the data associated with the tail labels.

In our analysis of the dataset, we find that a total of 3,188 labels (68.1%) have less than 4 entries in the dataset, i.e., tail labels. As such, it may be challenging for machine learning algorithms to train good models that accurately predict many labels due to the low amount of training data for the tail labels. The distribution of the label occurrence of the dataset can be seen in Figure 4. Moreover, considering the large proportion of labels with limited examples (78.1% labels have four or less examples), prior studies [11, 29] have suggested that deep learning-based approaches may not be suitable due to data scarcity and the lack of training data for the tail labels. However, our experiments have been surprising; the results of our experiments show that recent models, including the deep learning-based LightXML, achieve high effectiveness. This suggests that the problem of tail labels did not hinder the XML models in our task of identifying relevant libraries.

To understand this phenomenon, we analyze a subset of data where the labels occur four or less times, i.e., the tail labels, in the dataset. We find that vulnerabilities associated with libraries that appear four or less times in the dataset are more likely to have a vulnerability description that explicitly mentions at least one of the affected library names, which can be learned as highly discriminative features by the XML models. Out of 2,642 vulnerabilities, 2,191 (83%) of them contain an explicit mention of the library in the vulnerability report. This is a higher proportion than the libraries that appear more than four times; out of 5,024 vulnerabilities, 3,726 (74%) of them contain an explicit mention of the library. Using a Chi-Square test to compare the distributions, we obtain a Chi-Square statistic of over 75.5 and p-value less than 0.05, indicating that the distributions of library mentions of frequently appearing libraries differ from that of libraries uncommonly seen in the dataset. In other words, library names are more likely to appear in the text of vulnerability reports describing uncommon libraries.

Still, if a machine learning approach, which typically requires many examples for training [15, 41], does not encounter a label in the training dataset, it cannot correctly predict the label when the model is deployed. As such, an XML model is limited by its inability to correctly predict a previously unseen library affected by a vulnerability. This is a challenge that should be addressed by future work.

On how LightXML achieves better prediction accuracy. We conduct a deeper analysis on the prediction results of the XML models to get a better understanding of the more accurate predictions provided by the recent XML models. In particular, we compare the predictions of the baseline FastXML model against the best performing LightXML model. Figure 5 shows a Venn diagram comparing the top-1 predictions of both models. There are 164 vulnerability reports where LightXML provides a correct prediction while FastXML does not. On the other hand, there are 41 vulnerability reports where FastXML provides a correct prediction while LightXML does not. We further analyze these cases to get a better understanding of the XML models.

First, we analyze the 164 vulnerability reports for which the affected libraries are correctly predicted only by LightXML. We find that FastXML produces frequently occurring labels for the 164 vulnerability reports. FastXML incorrectly predicts 49 (30%) of them with `ImageMagick` library, 15 (9%) of them with `firefox` library, and 15 (9%) of them with `kernel-rt` library. These three libraries are some of the most frequently-occurring libraries within our dataset, with `ImageMagick`, `firefox`, and `kernel-rt` occurring in 5.2% (the most common label), 5.1% (the second most common label), and 3.8% (the fourth most common label) of the CVE entries respectively. This may indicate that FastXML defaulted to predicting the most frequent libraries on these vulnerability reports, which suggests that it did not manage to extract discriminating features from them.

Then, we analyze the 41 entries that are correctly predicted by FastXML but not by LightXML. We observe that most of the failures are caused by LightXML predicting other libraries that are related to the actual affected library. Consider CVE-2014-6468 and CVE-2015-0437. Both vulnerabilities affect `java-1.8.0-openjdk`. However, LightXML predicts `java-1.6.0-ibm`, which is a different version of the `java` library, instead. These cases account for 16 of the 41 (39%) failed prediction cases. We further analyze the 194 vulnerability reports where LightXML is unable to provide a correct top-1 prediction. We find a similar finding, where 93 out of the 194 (48%) incorrect top-1 predictions are due to LightXML predicting related libraries.

5.2 Future Directions

While using more sophisticated XML techniques have led to improvements, there are still vulnerability reports with affected libraries that were not correctly predicted by the techniques. We outline challenges and directions for future work:

Similar features with different labels. The features used for classification are extracted from the description, reference URLs, and CPE configuration of the vulnerability reports. We find that there are vulnerability reports with similar features but have different labels. An example is the vulnerability report of CVE-2014-1568. This vulnerability affects three libraries, namely *nss*, *nss-softokn*, and *nss-util*. However, LightXML and Bonsai identify *firefox*, *thunderbird*, and *nss* as the 3 most related labels. Upon investigation, we observe that the features of the report are similar to other reports of vulnerabilities affecting *firefox* and *thunderbird*. To check this hypothesis, we extract the term frequency - inverse document frequency (TF-IDF) vectors from our dataset using Scikit-Learn [36] gopivotal resources directory traversal vulnerability resources grails attackers information vectors issue split cve vulnerability types securityfocus http archives archives fulldisclosure html http www com security cve 0053

Figure 6: Extracted feature for CVE-2014-2858

library.[3] Then, we compare the similarity of CVE-2014-1568's vectors with other vulnerability reports' vectors. We compute the cosine similarity of two vectors and rank the reports by their similarity. We find that the features that are most similar to this report are taken from CVE-2014-1574 and CVE-2014-1590 with 0.70 cosine similarity. Both vulnerabilities affect *firefox* and *thunderbird*. Since the vulnerability reports have similar features, the XML models are unable to distinguish between their labels.

One solution to mitigate this problem is to incorporate other sources of data to extract features from. The above example suggests that the vulnerability report alone may not be informative enough to distinguish between vulnerability reports of different libraries, pointing at the need to consider other data beyond the vulnerability report.

Lack of quality features. After analysis, we also find that some reports have more details (e.g., more reference URLs) and more verbose descriptions than others. When the vulnerability reports lack informative features, the XML model is unable to identify the correct library. An example of this is CVE-2014-2858. The features extracted from the vulnerability report are shown in Figure 6. None of the features provide enough information to correctly infer the affected libraries.

Similar to the problem of similar features with different labels (described in the prior point), we suggest considering other types of features extracted from other sources of data. A possible new source of information is to collect data from the webpages referenced by the reference URLs rather than processing the URLs as text.

Libraries that rarely appear in the training data. Data sparsity hinders the effectiveness of machine learning techniques. In machine learning, having more training data often improves accuracy and generalizability [4, 20]. To alleviate this problem, future work can explore techniques from the ML field that target the lack of data, such as the use of data augmentation [13, 50, 58] to create artificial data from existing data.

In our application of predicting relevant libraries from vulnerability reports, one potential challenge is that the models will encounter vulnerabilities affecting libraries that were not previously seen by the XML model. One possible direction is to build on our finding that vulnerability reports may include explicit mention of uncommon library names. This challenge is related to the problem of "Out-of-Vocabulary" words [25]. A solution is to use techniques from the Natural Language Processing (NLP) domain, such as the Copy mechanism [18] from recent Deep Learning models, which learns to repeat important terms from the input text (in our case, the vulnerability report) even when the term has not been seen before.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

5.3 Threats to Validity

Threats to internal validity. Threats to internal validity relate to the possibility of errors in our implementation. To mitigate this risk, we have provided the detailed training parameters of the XML models in our experiments. We have also made our code publicly available in the following link: https://github.com/automated-library/ICPC_2022_Automated-Identification-of-Libraries-from-Vulnerability-Data Using the parameters and the available code, other researchers can replicate our work and confirm our findings.

Threats to construct validity. Threats to construct validity relate to the suitability of our evaluation metrics. For performance evaluation, the metrics that we use are precision, recall, and F1-score from the top-k prediction (k=1,2,3). These metrics are the same as used in prior work [11]. For the efficiency evaluation, we use the same metrics, the training and prediction time of the XML models, as prior works on XML [23, 29, 39]. As we use the same metrics as prior works, we believe that this threat is minimal.

Threats to external validity. Threats to external validity relate to the generalizability of our findings. In this study, we utilize the dataset by the prior study of Chen et al. [11], which contains 7,696 vulnerability reports and their labels. These vulnerability reports were curated and verified by security researchers in Veracode. Moreover, the reports are collected from NVD entries over a long time period between 2002 to 2019. Thus, we believe that this threat is minimal.

6 CONCLUSION AND FUTURE WORK

An essential part of software composition analysis (SCA) is the identification of the relevant libraries from a vulnerability report, which may not explicitly indicate them. A previous study has framed the problem as an extreme multi-label classification problem, in which machine learning approaches are used to predict affected libraries from the reports. We assess the effectiveness and efficiency of six XML models. These models are chosen as they have exhibited improved performance over FastXML, a baseline proposed in prior work, in other XML tasks.

We find that all models, including deep learning-based models, were effective. Specifically, we find that the Bonsai tree-based model and the LightXML transformer-based model achieve 7% and 10% average F1-score improvements over the baseline FastXML model respectively. We find that all XML models are highly practical, as each vulnerability report can be predicted in less than 100 milliseconds. Apart from the transformer-based LightXML, all models improve over FastXML in training and prediction time, with Bonsai and Parabel achieving the biggest improvement of 627x and 589x for training time, and 12x faster prediction time.

We analyze the performance of the models and discuss challenges and future directions regarding the use of XML on this task. Many vulnerability reports of different libraries do not contain sufficient detail to be distinguished from one another, suggesting that future work explore other sources of information beyond NVD.

REFERENCES

[1] Mohammed Ali Al-Garadi, Y.-C. Yang, H. Cai, Yucheng Ruan, Karen O'Connor, Graciela Gonzalez-Hernandez, Jeanmarie Perrone, and A. Sarker. 2020. Text Classification Models for the Automatic Detection of Nonmedical Prescription Medication Use from Social Media. *medRxiv* (2020).

[2] Mahmoud Alfadel, Diego Elias Costa, and Emad Shihab. 2021. Empirical Analysis of Security Vulnerabilities in Python Packages. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 446–457. https://doi.org/10.1109/SANER50967.2021.00048

[3] Rohit Babbar and Bernhard Schölkopf. 2017. DiSMEC: Distributed Sparse Machines for Extreme Multi-Label Classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (Cambridge, United Kingdom) *(WSDM '17)*. Association for Computing Machinery, New York, NY, USA, 721–729. https://doi.org/10.1145/3018661.3018741

[4] Michele Banko and Eric Brill. 2001. Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (Toulouse, France) *(ACL '01)*. Association for Computational Linguistics, USA, 26–33. https://doi.org/10.3115/1073012.1073017

[5] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse Local Embeddings for Extreme Multi-Label Classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 730–738.

[6] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python* (1st ed.). O'Reilly Media, Inc.

[7] Andrew Buttner, Todd Wittbold, and Neal Ziring. 2007. *Common Platform Enumeration (CPE)-Name Format and Description.* Technical Report. NATIONAL SECURITY AGENCY/CENTRAL SECURITY SERVICE FORT MEADE MD FORT MEADE ....

[8] Robit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. 2001. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[9] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3163–3171. https://doi.org/10.1145/3394486.3403368

[10] Francisco Charte, Antonio Rivera, María José del Jesus, and Francisco Herrera. 2016. On the impact of dataset complexity and sampling strategy in multilabel classifiers performance. In *International conference on hybrid artificial intelligence systems*. Springer, 500–511.

[11] Yang Chen, Andrew E. Santosa, Asankhaya Sharma, and David Lo. 2020. Automated Identification of Libraries from Vulnerability Data *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 90–99. https://doi.org/10.1145/3377813.3381360

[12] Yang Chen, Andrew E. Santosa, Ang Ming Yi, Abhishek Sharma, Asankhaya Sharma, and David Lo. 2020. A Machine Learning Approach for Vulnerability Curation. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 32–42. https://doi.org/10.1145/3379597.3387461

[13] Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. 2015. Data Augmentation for Deep Neural Network Acoustic Modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23, 9 (2015), 1469–1477. https://doi.org/10.1109/TASLP.2015.2438544

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.

[15] Simon Shaolei Du, Yining Wang, Xiyu Zhai, Sivaraman Balakrishnan, Ruslan Salakhutdinov, and Aarti Singh. 2018. How Many Samples are Needed to Estimate a Convolutional Neural Network?. In *NeurIPS*.

[16] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Y. Yang, Deming Chen, Marianne Winslett, Hassan Sajjad, and Preslav Nakov. 2021. Compressing Large-Scale Transformer-Based Models: A Case Study on BERT. *Transactions of the Association for Computational Linguistics* 9 (2021), 1061–1080.

[17] Antonios Gkortzis, Daniel Feitosa, and Diomidis Spinellis. 2021. Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. *Journal of Systems and Software* 172 (2021), 110653. https://doi.org/10.1016/j.jss.2020.110653

[18] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1631–1640.

[19] Nilesh Gupta, Sakina Bohra, Yashoteja Prabhu, Saurabh Purohit, and Manik Varma. 2021. Generalized Zero-Shot Extreme Multi-Label Learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 527–535. https://doi.org/10.1145/3447548.3467426

[20] Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The Unreasonable Effectiveness of Data. *Intelligent Systems, IEEE* 24 (05 2009), 8 – 12. https://doi.org/10.1109/MIS.2009.36

[21] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme Multi-Label Loss Functions for Recommendation, Tagging, Ranking Other Missing Label Applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 935–944.

[22] Kalina Jasinska, Krzysztof Dembczynski, Róbert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hüllermeier. 2016. Extreme F-Measure Maximization Using Sparse Probability Estimates. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (New York, NY, USA) *(ICML'16)*. JMLR.org, 1435–1444.

[23] Ting Jiang, Deqing Wang, Leilei Sun, Huayi Yang, Zhengyang Zhao, and Fuzhen Zhuang. 2021. LightXML: Transformer with Dynamic Negative Sampling for High-Performance Extreme Multi-label Text Classification. *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (May 2021), 7987–7994.

[24] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, Valencia, Spain, 427–431.

[25] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. 2020. Big code!= big vocabulary: Open-vocabulary models for source code. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1073–1085.

[26] Sujay Khandagale, Han Xiao, and Rohit Babbar. 2020. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning* 109 (11 2020). https://doi.org/10.1007/s10994-020-05888-2

[27] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[28] Qiang Li, Jinke Song, Dawei Tan, Haining Wang, and Jiqiang Liu. 2021. PDGraph: A Large-Scale Empirical Study on Project Dependency of Security Vulnerabilities. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 161–173. https://doi.org/10.1109/DSN48987.2021.00031

[29] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep Learning for Extreme Multi-Label Text Classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Shinjuku, Tokyo, Japan) *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 115–124. https://doi.org/10.1145/3077136.3080834

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv* abs/1907.11692 (2019).

[31] Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Yaming Yang, Quanlu Zhang, Yunhai Tong, and Jing Bai. 2020. LadaBERT: Lightweight Adaptation of BERT through Hybrid Model Compression. In *COLING*.

[32] Mark Marsden, Kevin McGuinness, Joseph Antony, Haolin Wei, Milan D. Redzic, Jian Tang, Zhilan Hu, Alan F. Smeaton, and Noel E. O'Connor. 2020. Investigating Class-Level Difficulty Factors In Multi-Label Classification Problems. *2020 IEEE International Conference on Multimedia and Expo (ICME)* (2020), 1–6.

[33] Eneldo Loza Mencía and Johannes Fürnkranz. 2008. Efficient Pairwise Multilabel Classification for Large-Scale Problems in the Legal Domain. In *ECML/PKDD*.

[34] Na Meng, Stefan Nagy, Danfeng (Daphne) Yao, Wenjie Zhuang, and Gustavo Arango Argoty. 2018. Secure Coding Practices in Java: Challenges and Vulnerabilities. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 372–383. https://doi.org/10.1145/3180155.3180201

[35] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. 2021. Deep Learning–Based Text Classification: A Comprehensive Review. *ACM Comput. Surv.* 54, 3, Article 62 (April 2021), 40 pages.

[36] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[37] Nam H. Pham, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. 2010. Detection of Recurring Software Vulnerabilities. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (Antwerp, Belgium) *(ASE '10)*. Association for Computing Machinery, New York, NY, USA, 447–456. https://doi.org/10.1145/1858996.1859089

[38] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. 2020. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering* (2020), 1–41.

[39] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising. In *WWW '18: Proceedings of the 2018 World Wide Web Conference*. 993–1002. https://doi.org/10.1145/3178876.3185998

[40] Yashoteja Prabhu and Manik Varma. 2014. FastXML: a fast, accurate and stable tree-classifier for extreme multi-label learning. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2014).

[41] Franco Scarselli, Ah Chung Tsoi, and Markus Hagenbuchner. 2018. The Vapnik–Chervonenkis dimension of graph and recursive neural networks. *Neural Networks* 108 (2018), 248–259. https://doi.org/10.1016/j.neunet.2018.08.010

[42] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2011. On the stratification of multi-label data. *Machine Learning and Knowledge Discovery in Databases* (2011), 145–158.

[43] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2011. On the stratification of multi-label data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 145–158.

[44] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. *ArXiv* abs/2004.02984 (2020).

[45] Piotr Szymański and Tomasz Kajdanowicz. 2017. A Network Perspective on Stratification of Multi-Label Data. In *Proceedings of the First International Workshop on Learning with Imbalanced Domains: Theory and Applications (Proceedings of Machine Learning Research, Vol. 74)*, Luís Torgo, Bartosz Krawczyk, Paula Branco, and Nuno Moniz (Eds.). PMLR, ECML-PKDD, Skopje, Macedonia, 22–35.

[46] Yukihiro Tagami. 2017. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-Label Classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 455–464. https://doi.org/10.1145/3097983.3097987

[47] Morteza Verdi, Ashkan Sami, Jafar Akhondali, Foutse Khomh, Gias Uddin, and Alireza Karami Motlagh. 2020. An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.3023664

[48] Tong Wei and Yu-Feng Li. 2019. Does Tail Label Help for Large-Scale Multi-Label Learning? *IEEE transactions on neural networks and learning systems* 31, 7 (2019), 2315–2324.

[49] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019).

[50] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. 2016. Understanding Data Augmentation for Classification: When to Warp?. In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 1–6. https://doi.org/10.1109/DICTA.2016.7797091

[51] Tong Wu, Qingqiu Huang, Ziwei Liu, Yu Wang, and Dahua Lin. 2020. Distribution-Balanced Loss for Multi-Label Classification in Long-Tailed Datasets. In *ECCV*.

[52] Marek Wydmuch, Kalina Jasinska, Mikhail Kuznetsov, Róbert Busa-Fekete, and Krzysztof Dembczyński. 2018. A No-Regret Generalization of Hierarchical Softmax to Extreme Multi-Label Classification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 6358–6368.

[53] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS*.

[54] Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. 2021. Pretrained Transformers for Text Ranking: BERT and Beyond. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (Virtual Event, Israel) *(WSDM '21)*. Association for Computing Machinery, New York, NY, USA, 1154–1156.

[55] Ian E.H. Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. 2017. PPDsparse: A Parallel Primal-Dual Sparse Method for Extreme Classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 545–553. https://doi.org/10.1145/3097983.3098083

[56] Ian E. H. Yen, Xiangru Huang, Kai Zhong, Pradeep Ravikumar, and Inderjit S. Dhillon. 2016. PD-Sparse: A Primal and Dual Sparse Approach to Extreme Multiclass and Multilabel Classification. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (New York, NY, USA) *(ICML'16)*. JMLR.org, 3069–3077.

[57] Ronghui You, Zihan Zhang, Ziye Wang, Suyang Dai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2019. AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification. In *NeurIPS*.

[58] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2017. Random Erasing Data Augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (08 2017). https://doi.org/10.1609/aaai.v34i07.7000