Exploiting Token and Path-based Representations of Code for Identifying Security-Relevant Commits

by

Achyudh Ram Keshav Ram

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Achyudh Ram Keshav Ram was the sole author for Chapters 1, 3, and 4 which were written under the supervision of Meiyappan Nagappan and Jimmy Lin.

This thesis consists in part of two manuscripts written for publication:

The research presented in Chapter 2 was co-authored with Ashutosh Adhikari and Raphael Tang, under the supervision of Jimmy Lin, and was accepted to NAACL-HLT 2018 [1].

The research presented in Chapter 3 was conducted with SAP Security Research and is currently under peer-review.

Abstract

Public vulnerability databases such as CVE and NVD account for only 60% of security vulnerabilities present in open-source projects and are known to suffer from inconsistent quality. Over the last two years, there has been considerable growth in the number of known vulnerabilities across projects available in various repositories such as NPM and Maven Central. However, public vulnerability management databases such as NVD suffer from poor coverage and are too slow to add new vulnerabilities. Such an increasing risk calls for a mechanism to promptly infer the presence of security threats in open-source projects. In this thesis, we seek to address this problem by treating the identification of security-relevant commits as a classification task.

Since existing literature on neural networks for commit classification is sparse, we first turn to document classification for inspiration. Extensive research in this domain, on the other hand, has resulted in increasingly complex neural models, with a number of researchers questioning the necessity of such architectures. We conduct a large-scale reproducibility study of several recent neural network models, and show that well-executed, simpler models are quite effective for document classification. We find that a simple bi-directional LSTM with regularization yields competitive accuracy and F_1 on four benchmark document classification datasets.

Based on trends in document classification and the domain-specific peculiarities of commit classification, we build a family of hierarchical neural network models for the identification of security-relevant commits. We evaluate five different input representations and show that models that learn on tokens extracted from the commit diff are simpler and more effective than models that learn from path-contexts extracted from the AST. We also show that providing the models with contextual information through features extracted from the source code improves accuracy and F_1 further, and discuss why path-based models might not capture any additional information compared to token-based models for this task. Finally, we make a case for reporting standard deviation of test scores across multiple runs in order to avoid erroneous conclusions and establish robust baselines.

Acknowledgements

I would like to thank my advisors Professor Jimmy Lin and Professor Meiyappan Nagappan for their support and guidance. They provided me the opportunity to work on fascinating problems over the past two years and this thesis would not have been possible without them. I would also like to thank the readers of my thesis, Professor Mike Godfrey and Professor Yaoliang Yu, for taking the time to review my work.

Finally, I am grateful to the members of the Data Systems Group and the Software Analytics Group — many of whom friends and collaborators — for making my time at the University of Waterloo memorable.

Dedication

This thesis is dedicated to my parents for their boundless encouragement and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of open-source software has been steadily increasing over the past couple of decades, with the number of Java packages in Maven Central doubling in 2018. Meanwhile, Tal [45] states that there has been an 88% growth in the number of vulnerabilities reported in open source software over the last two years. In order to develop secure software, it is essential to analyze and understand security vulnerabilities that occur in software systems and address them in a timely manner.

Common Vulnerabilities and Exposures (CVE)[1] is a list of publicly known cybersecurity vulnerabilities, each with an identification number. These entries are used in the National Vulnerability Database (NVD)[2], the U.S. government repository of standards-based vulnerability management data. NVD suffers from poor coverage, as it contains only 10% of the open-source vulnerabilities that have received a CVE identifier [42]. This could be due to a number of security vulnerabilities being discovered and fixed through informal communication channels between maintainers and their users in an issue tracker. To make things worse, these public databases are too slow to add vulnerabilities as they lag behind private databases such as Snyk's DB by an average of 92 days [45]. All of the above pitfalls of public vulnerability management databases call for a mechanism to automatically infer the presence of security threats in open-source projects, and the corresponding fixes, in a timely manner.

While there exist various approaches in the literature for identifying and managing security vulnerabilities, Ponta et al. [37] argue that an effective vulnerability management approach must be code centric; rather than relying on metadata, vulnerabilities and their fixes must be analyzed at the code level. We take this approach.

[1] https://cve.mitre.org/ [2] https://nvd.nist.gov/

In this thesis, we treat the identification of security-relevant commits as a binary document classification problem by considering the code present in commits as documents. Given the extensive literature in the natural language processing (NLP) community on applying neural networks for document classification, we first perform a large-scale reproducibility study of several recent neural network models. We then combine the lessons learned from the reproducibility study and domain-specific knowledge on commit classification to build a family of hierarchical neural network models for the identification of security-relevant commits.

1.1 Neural Networks for Document Classification

Since existing literature on neural networks for commit classification is sparse, we turn to document classification for inspiration. We begin this thesis with an exploration into neural networks for document classification. While this topic has been studied in far more detail by the NLP community, it has also led to the development of overly complex neural architectures and modeling techniques.

Worryingly, these new models are accompanied by smaller and smaller improvements in effectiveness on standard benchmark datasets, which leads us to wonder if observed improvements are "real". There is, however, ample evidence to the contrary. Sculley et al. [43] lament the lack of empirical rigor in the machine learning community and cite examples where improvements can be attributed to far more mundane reasons (e.g., hyperparameter tuning) or are simply noise. Lipton and Steinhardt [26] concur with these sentiments, adding that authors often use fancy mathematics to obfuscate or to impress (reviewers) rather than to clarify. Further, Crane [11] talk about the growing concern within the NLP community about the reproducibility of results. Complex architectures are more difficult to train, more sensitive to hyperparameters, and brittle with respect to domains with different data characteristics—thus both exacerbating the "crisis of reproducibility" and making it difficult for practitioners to deploy networks that tackle real-world problems in production environments.

We conduct a large-scale reproducibility study of several recent neural models and find that a simple bi-directional LSTM (BiLSTM) architecture with appropriate regularization yields accuracy and F_1 scores that are either competitive or exceed the state of the art on four standard benchmark datasets. While these regularization techniques, borrowed from language modeling, are not novel, we are to our knowledge the first to apply them in this context. This work provides an open-source platform for future work in document

classification and the foundation upon which we build neural models for the identification of security-relevant commits [1].

1.2 Identifying Security-Relevant Commits

Building on our reproducibility-study on document classification, we apply some of the lessons learned to develop neural models for commit classification. We propose a family of hierarchical neural network models for the identification of security-relevant commits and the evaluation of different input representations for commit classification. These models share a hierarchical structure that uses convolutional layers to progressively build a commit-level feature vector from tokens extracted from either the commit diff or from source code.

We compare the effectiveness of five input representations:

1. **Bag of words:** a multiset of the tokens present in the commit diff, after tokenizing it with a lexer, from which tf–idf features are extracted.
2. **Bag of paths:** a multiset of the path-context vectors extracted from the abstract syntax tree (AST); we go into more detail in Section 3.2.
3. **Commit diff tokens:** a sequence of tokens extracted from the commit diff with a lexer; we remove comments from the code as may point out any vulnerabilities present or fixed.
4. **Paired source code tokens:** a pair of two token sequences extracted from the source code before and after a commit. This would have more contextual information than a diff token sequence as it contains the source code for entire Java classes, rather than just the modified chunks.
5. **Hybrid:** a combination of commit diff tokens and bag of paths. Training a model on features extracted from both the commit diff and path-contexts allows us to examine if these models are capturing different signals.

We collectively refer to models that learn from bag of paths as path-based models, and bag of words, commit diff tokens, or paired source code tokens as token-based models.

Our results show models that learn from commit diff tokens are more effective than models that learn from path-contexts extracted from the AST. In fact, Code2Vec, claimed by Alon et al. [5] to be suitable for a wide range of source code classification tasks, performs

worse than an SVM baseline. Furthermore, we show that providing the models with additional contextual information by training them on paired source code tokens improves their performance further. We also discuss why path-based models might not capture any additional information compared to token-based models for this task, with a hybrid model that learns from a combination of two input representations: diff tokens and bag of paths.

As studies that use neural networks become more prevalent, the software engineering (SE) community will also have to contend with the "crisis of reproducibility" discussed in Section 1.1. A number of studies argue that a single performance score is insufficient to compare non-deterministic approaches and emphasize reporting the standard deviation of test scores across multiple runs to ensure the stability of the results and avoid erroneous conclusions [40, 11]. We concur with this sentiment and in order to establish robust baselines, we compare score distributions based on multiple runs with different random seeds.

All of our models can be used for inference on large repositories without having to go through a time consuming compilation process. We envision that this work would ultimately allow practitioners to build a repository of potential vulnerability fixes by monitoring open-source projects in real-time and detecting security-relevant changes.

1.3 Thesis Statement

Simple neural models adapted from document classification are effective in identifying security-relevant commits from just the tokens in the commit diff.

1.4 Contributions

The main contributions of this thesis can be summarized as follows:

- A comprehensive reproducibility study of several neural network models on four benchmark document classification datasets.
- A comparative analysis of the effectiveness of several neural network models, spanning five different input representations, for the identification of security-relevant commits.
- An evaluation framework for comparing models by treating performance metrics as distributions rather than individual data points; this can be used in future SE studies that rely on a non-deterministic approach.

- Two open-source libraries that can serve as the foundation for future work:
  - Hedwig, which contains a range of document classification models implemented from scratch: https://github.com/castorini/hedwig
  - Omniocular, which contains our hierarchical neural network models and baselines for the identification of security-relevant commits: https://github.com/omniocular/omniocular

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

- In Chapter 2, Section 2.1 introduces the research questions we seek to answer in our reproducibility study; Section 2.2 reviews related work in document classification and regularization; Section 2.3 details the architecture of our regularized BiLSTM models; Section 2.4 discusses the experimental setup, datasets and the hyperparameter configuration; Section 2.5 presents the results of the replication study on four standard benchmark datasets.
- In Chapter 3, Section 3.1 introduces the research questions we seek to answer in our analysis; Section 3.2 reviews related work on distributed representations for source code and the identification of security vulnerabilities; Section 3.3 outlines the experimental setup and dataset; Section 3.4 details our methodology for the identification of security-relevant commits, the baselines used for comparison and the architecture of our hierarchical models; Section 3.5 presents the results of our approach and further analysis on five different input representations for commit classification; Section 3.6 discusses the threats to validity of our approach.
- Chapter 4 concludes this thesis, summarizes the main contributions and discusses future work.

Chapter 2

Neural Networks for Document Classification

In this chapter, we conduct a large-scale reproducibility study of several recent neural models on four standard benchmark datasets. We describe a simple BiLSTM model, and examine the effectiveness of regularization techniques such as embedding dropout and weight dropping for document classification. This work is reported in Adhikari* et al. [1].

2.1 Research Questions

Through this reproducibility study, we seek to answer the following research questions:

- **RQ1:** *Are neural models from recent document classification literature more effective than a simple BiLSTM?* We reimplement a range of neural models from recent literature and compare their effectiveness with a simple BiLSTM baseline on four standard benchmark datasets.
- **RQ2:** *Does regularization improve the effectiveness of our BiLSTM model for document classification?* We examine the impact of regularization techniques borrowed from language modeling on the effectiveness of our BiLSTM model.

2.2 Background and Related Work

2.2.1 Document Classification

Over the last few years, deep neural networks have achieved the state of the art in document classification. Kim [20] show that convolutional neural networks (CNNs) can effectively perform single-sentence sentiment prediction, among other sentence classification tasks. In this approach, the vector representations of the words in a sentence are concatenated vertically to create a two-dimensional matrix for each sentence. The resulting matrix is passed through a CNN to extract higher-level features for performing the classification.

Yang et al. [51] introduce the hierarchical attention network (HAN), where a document vector is progressively built with word- and sentence-level attention by first aggregating important words into sentence vectors, and then aggregating important sentences vectors into document vectors. Although this model nicely captures the intuition that modeling word sequences in sentences should be handled separately from sentence-level discourse modeling, one wonders if such complex architectures are really necessary for document classification, especially given the size of training data available for this task.

An important variant of document classification is the multi-label, multi-class case. Liu et al. [27] develop XML-CNNs for multi-label text classification, basing the architecture on KimCNN [20] with increased filter sizes and an additional fully-connected layer. They also incorporate dynamic adaptive max-pooling [10] instead of the vanilla max-pooling over time in KimCNN. The paper compares with CNN-based approaches for the multi-label task, but only reports precision and disregards recall. Yang et al. [50] instead adopts encoder–decoder sequence generation models (SGMs) for generating multiple labels for each document. Similar to our critique of HAN, we opine against the high complexity of these multi-label approaches.

2.2.2 Regularization

Deep neural networks are prone to overfitting due to the possibility of the network learning complicated relationships that exist in the training set but not in unseen test data. Dropout prevents complex co-adaptations of hidden units on training data by randomly removing—i.e., dropping out hidden units along with their connections during training [44].

There have been attempts to extend dropout [44] from feedforward neural networks to recurrent ones. Unfortunately, direct application of dropout on the hidden units of an

RNN empirically harms its ability to retain long-term information [52]. Recently, however, Merity et al. [28] successfully apply dropout-like techniques to regularize RNNs for language modeling, and show that this reduces word-level perplexity on multiple datasets. Inspired by this development, we adopt two of their regularization techniques, embedding dropout and weight-dropped LSTMs, to our task of document classification. We also adapt DropBlock, a regularization technique for CNNs, to our hierarchical models in Chapter 3.

Weight-dropped LSTM. LSTMs comprise eight total input–hidden and hidden–hidden weight matrices; in weight dropping, Merity et al. [28] regularize the four hidden–hidden matrices with DropConnect [47]. The operation is applied only once per sequence, using the same dropout mask across multiple timesteps. Conveniently, this allows practitioners to use fast, out-of-the-box LSTM implementations without affecting the RNN formulation or training performance.

Embedding Dropout. Introduced in Gal and Ghahramani [15] and successfully employed for neural language modeling [28], embedding dropout performs dropout on entire word embeddings, effectively removing some of the words at each training iteration. As a result, the technique conditions the model to be robust against missing input; for document classification, this discourages the model from relying on a small set of input words for prediction.

DropBlock. While dropout works well for regularizing fully connected layers, it is less effective for convolutional layers due to the spatial correlation of activation units in convolutional layers. There have been a number of attempts to extend dropout to convolutional neural networks [48]. DropBlock is a form of structured dropout for convolutional layers where units in a contiguous region of a feature map are dropped together [16].

2.3 Model

We design our model to be minimalistic: First, we feed the word embeddings $\mathbf{w}_{1:n}$ of a document to a single-layer BiLSTM, extracting concatenated forward and backward word-level context vectors $\mathbf{h}_{1:n} = \mathbf{h}^f_{1:n} \oplus \mathbf{h}^b_{1:n}$. Subsequently, we max-pool $\mathbf{h}_{1:n}$ across time to yield document vector $\mathbf{d}$—see Figure 2.1, labels a–f. Finally, we feed $\mathbf{d}$ to a sigmoid or a softmax layer over the labels, depending on if the task type is multi-label or single-label classification (label g).

Contrary to prior art, our approach refrains from attention, hierarchical structure, and sequence generation, each of which increases model complexity. For one, hierarchical structure requires sentence-level tokenization and multiple RNNs. For another, sequence

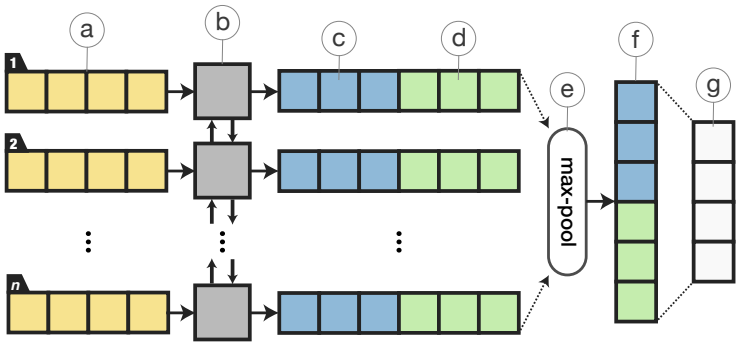


Figure 2.1: Illustration of the model architecture, where the labels are the following: (a) input word embeddings (b) BiLSTM (c, d) concatenated forward $h^f_{1:n}$ and backward $h^b_{1:n}$ hidden features (e) max-pooling over time (f) document feature vector (g) softmax or sigmoid output.

generation uses an encoder–decoder architecture, reducing computational parallelism. All three methods add complexity to the model; our approach instead uses a single-layer BiLSTM with trivial max-pooling and concatenation operations, which makes for both simple implementation and resource-efficient inference.

2.4 Experimental Setup

We conduct a large-scale reproducibility study involving HAN, XML-CNN, KimCNN, and SGM. These are compared to our proposed model, referred to as R-LSTM$_{\text{TEXT}}$, as well as an ablated variant without regularization, denoted LSTM$_{\text{TEXT}}$. The implementation of our model as well as from-scratch reimplementations of all the comparison models (except for SGM) are provided in our toolkit called Hedwig, which we make publicly available to serve as the foundation for future work.[1] In addition, we compare the neural approaches to logistic regression (LR) and support vector machines (SVMs). The LR model is trained using a one-vs-rest multi-label objective, while the SVM is trained with a linear kernel. We chose a linear kernel as we did not see any noticeable improvement in performance with a radial basis function kernel. Both of these models use word-level tf–idf vectors of the documents as features.

All of our experiments are performed on Nvidia GTX 1080 and RTX 2080 Ti GPUs, with PyTorch 0.4.1 as the backend framework. We use Scikit-learn 0.19.2 for computing

[1] http://hedwig.ca

Dataset	No. of classes	No. of samples	W	S
Reuters	90	10,789	144.3	6.6
AAPD	54	55,840	167.3	1.0
IMDB	10	135,669	393.8	14.4
Yelp 2014	5	1,125,386	148.8	9.1

Table 2.1: Summary of the datasets. W and S denote the average number of words and sentences per document, respectively.

the tf–idf vectors and implementing LR and SVMs.

2.4.1 Datasets

We evaluate our models on the following four datasets: Reuters-21578, arXiv Abstract Paper dataset (AAPD), IMDB, and Yelp 2014. Reuters and AAPD are multi-label datasets, whereas IMDB and Yelp are single-label ones. For IMDB and Yelp, we use random sampling to split the dataset such that 80% is used for training, 10% for validation, and 10% for test. We use the standard ModApte splits [6] for the Reuters dataset, and author-defined splits for AAPD [50]. We summarize the statistics of these datasets in Table 2.1.

Unfortunately, there is little consensus within the natural language processing community for choosing the splits of IMDB and Yelp 2014. Furthermore, they are often unreported in modeling papers, hence preventing direct comparison with past results. We are not able to find the exact splits Yang et al. [51] use; for consistency, we use the same proportion the authors report, but of course this yields different samples in each split.

2.4.2 Training and Hyperparameters

To ensure a fair comparison, we tune the hyperparameters for all baseline models. For HAN, we use a batch size of 32 across all the datasets, with a learning rate of 0.01 for Reuters and 0.001 for the rest. To train XML-CNN, we select a dynamic pooling window length of eight, a learning rate of 0.001, and 128 output channels, with batch sizes of 32 and 64 for single-label and multi-label datasets, respectively. For KimCNN, we use a batch size of 64 with a learning rate of 0.01. For training SGM on Reuters, we use the source code provided by the authors[2] and follow the same hyperparameters in their paper [50].

[2] https://github.com/lancopku/SGM

#	Model	Reuters		AAPD		IMDB		Yelp '14	
		Val. F_1	Test F_1	Val. F_1	Test F_1	Val. Acc.	Test Acc.	Val. Acc.	Test Acc.
1	LR	77.0	74.8	67.1	64.9	43.1	43.4	61.1	60.9
2	SVM	89.1	86.1	71.1	69.1	42.5	42.4	59.7	59.6
3	KimCNN *R.*	83.5 ±0.4	80.8 ±0.3	54.5 ±1.4	51.4 ±1.3	42.9 ±0.3	42.7 ±0.4	66.5 ±0.1	66.1 ±0.6
4	KimCNN *O.*	–	–	–	–	–	37.6	–	61.0
5	XML-CNN *R.*	88.8 ±0.5	86.2 ±0.3	70.2 ±0.7	68.7 ±0.4	–	–	–	–
6	HAN *R.*	87.6 ±0.5	85.2 ±0.6	70.2 ±0.2	68.0 ±0.6	51.8 ±0.3	51.2 ±0.3	68.2 ±0.1	67.9 ±0.1
7	HAN *O.*	–	–	–	–	–	49.4	–	**70.5**
8	SGM *O.*	82.5 ±0.4	78.8 ±0.9	–	**71.0**	–	–	–	–
9	$\text{LSTM}_{\text{TEXT}}$	87.6 ±0.2	84.9 ±0.3	72.1 ±0.4	69.6 ±0.4	52.5 ±0.2	52.1 ±0.3	68.6 ±0.1	68.4 ±0.1
10	$\text{R-LSTM}_{\text{TEXT}}$	**89.1** ±0.8	**87.0** ±0.5	**73.1** ±0.4	70.5 ±0.5	**53.4** ±0.2	**52.8** ±0.3	**69.0** ±0.1	68.7 ±0.1

Table 2.2: Results for each model on the validation and test sets; best values are bolded. *O.* refers to point estimates copied from Tang et al. [46], Yang et al. [51] and Yang et al. [50]; *R.* reports mean ± SD of five runs from our re-implementations.

For the LR and SVM models, we use the default set of hyperparameters in Scikit-learn.

For $\text{R-LSTM}_{\text{TEXT}}$ and $\text{LSTM}_{\text{TEXT}}$, we use the Adam optimizer with a learning rate of 0.01 on Reuters and 0.001 on the rest of the datasets, using batch sizes of 32 and 64 for multi-label and single-label tasks, respectively. For $\text{R-LSTM}_{\text{TEXT}}$, we also apply temporal averaging (TA): as shown in Kingma and Ba [21], TA reduces both generalization error and stochastic noise in recent parameter estimates from stochastic approximation. We set the default TA exponential smoothing coefficient of β_{EMA} to 0.99. We choose 512 hidden units for the BiLSTM models, whose max-pooled output is regularized using a dropout rate of 0.5. We also regularize the input–hidden and hidden–hidden BiLSTM connections using embedding dropout and weight dropping, respectively, with dropout rates of 0.1 and 0.2.

For our optimization objective, we use cross-entropy and binary cross-entropy loss for single-label and multi-label tasks, respectively. On all datasets and models, we use 300-dimensional word vectors [30] pre-trained on Google News. We train all neural models for 30 epochs with five random seeds, reporting the mean validation set scores and their corresponding test set results.

2.5 Results and Discussion

We report the mean and standard deviation (SD) of the F_1 scores and accuracy for all five runs in Table 2.2. For HAN and KimCNN, we include results from the original papers to

validate our reimplementation. We fail to replicate the reported results of SGM on AAPD using the authors' codebase and data splits. As a result, we simply copy the value reported in Yang et al. [50] in Table 2.2, row 8, which represents their maximum F_1 score. To verify the correctness of our HAN and KimCNN reimplementations, we compare the differences in F_1 and accuracy on the appropriate datasets. We attribute the small differences to using different dataset splits (see Section 2.4.1) and reporting mean values.

Baseline Comparison. First off, we see that the non-neural LR and SVM baselines perform remarkably well. On Reuters, for example, the SVM beats many neural baselines, including our non-regularized $\text{LSTM}_{\text{TEXT}}$ (rows 2–9). On AAPD, the SVM either ties or beats the other models, losing only to SGM (rows 2–8). Compared to the SVM, the LR baseline appears better suited for the single-label datasets IMDB and Yelp 2014, where it achieves better accuracy than the SVM does. As the closest comparison point, we find no benefit to HAN as $\text{LSTM}_{\text{TEXT}}$ achieves just as good or better classification results without attention mechanisms (rows 6 and 9).

Our simple R-$\text{LSTM}_{\text{TEXT}}$ model achieves state of the art on Reuters and IMDB (Table 2.2, rows 9 and 10), establishing mean scores of 87.0 and 52.8 for F_1 score and accuracy on the test sets of Reuters and IMDB, respectively. This highlights the efficacy of proper regularization and optimization techniques for the task. We observe that R-$\text{LSTM}_{\text{TEXT}}$ consistently improves upon the accuracy and F_1 of $\text{LSTM}_{\text{TEXT}}$ across all of the tasks—see rows 9 and 10, where, on average, regularization yields increases of 1.5 and 0.5 points for F_1 score and accuracy, respectively.

A few of our R-$\text{LSTM}_{\text{TEXT}}$ runs attain state-of-the-art test F_1 scores on AAPD. However, in the interest of robustness, we report the mean value. We also find the accuracy of R-$\text{LSTM}_{\text{TEXT}}$ and our reimplemented version of HAN on Yelp 2014 to be almost two points lower than the copied result of HAN (rows 6, 7, and 10) from Yang et al. [51]. On the other hand, both of the models surpass the original result by nearly two points for the IMDB dataset. We cannot rule out that these disparities are caused by the absence of any widely-accepted splits for evaluation on Yelp 2014 and IMDB (as opposed to model or implementation differences).

Toward Robust Baselines. Recently, reproducibility is becoming a growing concern for the NLP community [11]. Indeed, very few of the papers that we consider in this study report results on multiple seeds We provide the standard deviation of the scores across different seeds to demonstrate the stability of our results; doing so is good practice, since it reinforces the validity of the experimental results and claims. This is in line with previous papers [53, 39, 11] that emphasize reporting variance for robustness against potentially spurious conclusions.

2.6 Conclusions

In this chapter, we question the complexity of existing neural network architectures for document classification. To demonstrate the effectiveness of proper regularization and optimization, we apply embedding dropout, weight dropping, and temporal averaging when training a simple BiLSTM model, establishing either competitive or state-of-the-art results on multiple datasets.

This reproducibility study provides the foundation upon which we build neural models for the identification of security-relevant commits. Not only are $\text{LSTM}_{\text{TEXT}}$ and R-$\text{LSTM}_{\text{TEXT}}$ excellent baselines to compare our models against, we can also take inspiration from the extensive literature on neural document classification models. Like documents, commits have a hierarchical structure: a commit contains changes to multiple files, which in turn contains multiple lines of tokens. We can exploit this structure to build hierarchical models similar to Yang et al. [51], while incorporating regularization techniques such as embedding dropout and DropBlock.

Chapter 3

Identifying Security-Relevant Commits

In this chapter, building on the insights from our initial study on document classification, we propose a family of hierarchical neural network models for the identification of security-relevant commits. While this hierarchical modelling is similar to the work of Yang et al. [51] in document classification, we eschew attention mechanisms to keep our architecture simple—we instead focus on adapting regularization techniques from Section 2.2. In the process, we compare the effectiveness of five different input representations for commit classification, and examine the impact of regularization on model quality.

3.1 Research Questions

We break down our analysis into five research questions:

- **RQ1:** *Are token-based models better than path-based models for identifying security-relevant commits?* We examine how effective models that do not have access to commit metadata or actual source code for training are at identifying security-relevant commits. We compare our results with Code2Vec trained on the symmetric difference of path-contexts of all Java classes before and after a commit [9].
- **RQ2:** *Does extracting file-level features before and after a commit improve the identification of security-relevant commits?* We investigate if providing the model with contextual information, by training it on paired source code tokens, allows us to

better identify security-relevant commits. We encode all Java classes before and after a commit with two separate encoders. We then concatenate the intermediate representations from both of these encoders to get the class probability scores.

- **RQ3:** *Do path-based and token-based models capture different signals when identifying security-relevant commits?* If we see an improvement in performance with a hybrid model that uses the features extracted from both the commit diff and path-contexts to calculate the class probability scores, then we can say that these models are capturing different signals. If we do not, path-based models potentially do not capture any additional information compared to token-based models.
- **RQ4:** *Does regularization improve the effectiveness of our models in identifying security-relevant commits?* In Chapter 2, we demonstrate the effectiveness of proper regularization for document classification. We employ regularization techniques discussed in 2.2 across all our models to see if it helps with generalization and consequently improve their effectiveness.

Answering RQ1, RQ2 and RQ3 would help researchers and practitioners choose an appropriate input representation and architecture for their models, while answering RQ4 would help in fine-tuning these model architectures for this task.

3.2 Background and Related Work

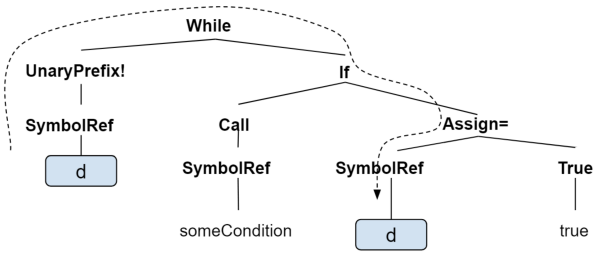
3.2.1 Distributed Representations for Source Code

In computational linguistics, there has been a lot of effort over the last few years to create a continuous higher dimensional vector space representation of words, sentences, and even documents such that similar entities are closer to each other in that space [29, 22, 23]. Mikolov et al. [29] introduced word2vec, a class of two-layer neural network models that are trained on a large corpus of text to produce word embeddings for natural language. Such learned distributed representations of words have accelerated the application of deep learning techniques for natural language processing tasks [7].

In the same vein as Mikolov et al. [29], neural networks have been used for representing snippets of code as continuous distributed vectors [5, 4]. The authors represent a path in the AST as a sequence of nodes connected by up and down arrows that symbolize the up or down link between adjacent nodes in the AST. Figure 3.1 shows one such possible path. This path representation is concatenated with the values of the leaf nodes the

```
1   while (!d) {
2       if (someCondition()) {
3           d = true;
4       }
5   }
6
```

(a) A simple Java code snippet



(b) The AST for the code snippet

Figure 3.1: A Java code snippet and its AST; the dotted line shows one of many possible paths along the AST. Figure adapted from Alon et al. [5]

path is connecting, which creates a path-context. A code snippet is a bag of its extracted path-contexts. Each element in the path-context is mapped to a distributed vector representation. These vector representations are jointly learnt with a path-attention network that aggregates these context vectors in a weighted manner.

While Code2Vec is the most popular, a number of other embedding techniques for source code are present in the literature. Henkel et al. [17] learn word embeddings from abstractions of traces obtained from the symbolic execution of a program. They evaluate their learned embeddings on a benchmark of API-usage analogies extracted from the Linux kernel and achieved 93% top-1 accuracy. Husain [18] describe a pipeline that leverages deep learning for semantic search of code. To achieve this, they train a sequence-to-sequence model that learns to summarize Python code by predicting the corresponding docstring from the code blob, and in the process provide code representations for Python.

While building usable embeddings for source code that capture the complex characteristics involving both syntax and semantics is a challenging task, such embeddings have direct downstream applications in tasks such as semantic code clone detection, code captioning, and code completion [3, 49].

3.2.2 Identifying Security Vulnerabilities

There exist a handful of papers in SE literature that perform commit classification to identify security vulnerabilities or fixes. Bosu et al. [8] conduct an analysis to identify which security vulnerabilities can be discovered during code review, or what characteristics of developers are likely to introduce vulnerabilities. Zhou and Sharma [54] describe a vulnerability identification system geared towards tracking large-scale projects in real time using

latent information underlying commit messages and bug reports in open-source projects. While Zhou and Sharma [54] classify commits based on the commit message, we use only the commit diff or the corresponding source code as features for our model.

Li et al. [24] describe a deep learning-based vulnerability detection system that seeks to improve the false negative rate by not requiring human experts to manually define features. Russell et al. [41] compile a dataset of potential security vulnerabilities based on the output of three different static analyzers, and use this dataset to evaluate a vulnerability detection tool that directly interprets source code using deep feature representation learning.

Closer to our work, Sabetta and Bezzi [42] propose a machine learning approach to identify security-relevant commits. Similar to our approach, they treat source code as documents written in natural language, but use well-known document classification methods to perform the actual classification. Cabrera-Lozoya et al. [9] study the impact of pre-training a variant of Code2Vec using two different pretext tasks versus a random initialization. They also claim that models trained on path-based representations outperform those that are trained on token-based representations, something that we'll explore further in this study.

3.3 Experimental Setup

All of the experiments are conducted on a system equipped with a Intel Core i7 7820X CPU and a pair of Nvidia GTX 2080 Ti GPUs. Our framework is written in Python 3.7, with our neural network models implemented in PyTorch 1.1.0 [32]. We make use of Scikit-learn 0.21.2 [33] for LR and SVM baselines, and TorchText 0.3.1 for implementing data preprocessing pipelines for token-based models.

We run our experiments with 50 random seeds and report the mean and standard deviation of the metrics across all the runs. In our experiments, random seeds control both the weight initialization for our models and the order in which data is presented to our model for training.

To extract token-level features for our model, we use the lexer and tokenizer provided as a part of the Python javalang library.[1] This allows us to ensure that we do not use code comments or metadata when constructing the inputs for our models as it is possible for comments or commit messages to include which vulnerabilities are fixed, as shown in Figure 3.2. Our models would then overfit on these features rather than actually learning to identify security vulnerabilities from code.

[1] https://pypi.org/project/javalang/

```java
/**
* Set the <code>acceptCookieNames</code> pattern of
* allowed names of cookies to protect against remote
* command execution vulnerability *
* @param pattern used to check cookie name against
*/
public void setAcceptCookieNames(String pattern) {
    acceptedPattern = Pattern.compile(pattern);
}
```

Figure 3.2: A code snippet from Apache Struts with the Javadoc stating which vulnerability was addressed

We use a manually-curated dataset of publicly disclosed vulnerabilities in 205 distinct open-source Java projects provided by Ponta et al. [36] as our positive training examples. We follow the same approach as Cabrera-Lozoya et al. [9] to generate negative training examples: for each commit in the above dataset, we pick a random commit from the same repository, under the assumption that security-relevant commits are few and far in-between. This results in a total of 1,950 commits, with equal number of positive and negative examples. We use random sampling to split the dataset such that 60% is used for training, and 20% each for validation and testing. In order to minimize the occurrence of duplicate commits in two of these splits (such as in both train and test), commits from no repository belong to more than one split.

We also compare the quality of models trained with randomly initialized and pre-trained embeddings. Since word2vec embeddings only need unlabelled data to train, the data collection and preprocessing stage is straightforward. GitHub, being a very large host of source code, contains enough code for training such models. However, a significant proportion of code in GitHub does not belong to engineered software projects [31]. To reduce the amount of noise in our training data, we filter repositories based on their size, commit history, number of issues, pull requests, and contributors, and build a corpus of the top 1000 Java repositories. We limit the number of repositories to 1000 due to GitHub API limitations. It is worth noting that using a larger training corpus might provide better results.

Similarly, we use the dataset provided by Allamanis et al. [2] to pre-train Code2Vec. This dataset contains around 700K examples in Java, split such that 9 projects are used for training, and 1 each for validation and testing.

3.4 Methodology

3.4.1 Pre-trained Embeddings

We learn token-level word2vec embeddings for code using the CBOW architecture [29] with negative sampling and a context window size of 5. We do not normalize variable identifiers into generic tokens as they can contain potentially useful contextual information. We perform minimal preprocessing on the input code sequence before pre-training the word2vec embeddings. This includes:

1. Removal of comments and whitespace when performing tokenization with a lexer.
2. Conversion of all numbers such as integers and floating point units into reserved tokens.
3. Removal of tokens whose length is greater than or equal to 64 characters.
4. Thresholding the size of the vocabulary to remove infrequent tokens.

We pre-train Code2Vec on over 700K Java methods using the semantic labelling task from Alon et al. [4], where the model is trained to predict a method's name from the body. While Cabrera-Lozoya et al. [9] claim that training on a more relevant pre-training task such as predicting the Jira ticket priority is more beneficial, we do not adopt this approach as their dataset is not publicly available.

3.4.2 Baselines

We benchmark the performance of our hierarchical models against Logistic Regression (LR) and Support Vector Machine (SVM) baselines that are trained on tf–idf features on the Java tokens extracted from the commit diffs.

For the bag of paths baseline, we adapt the Code2Vec model from Alon et al. [4] by replacing the last layer with a fully connected softmax layer that outputs a probability distribution over the security relevance of a commit. In order to train Code2Vec on commits, we follow an approach similar to Cabrera-Lozoya et al. [9]. We transform all Java classes before and after a commit into two separate sets of path-context vectors. We take the symmetric difference between these two sets in order to remove the unchanged contexts, and then feed the resulting set of path-contexts as input to the model.

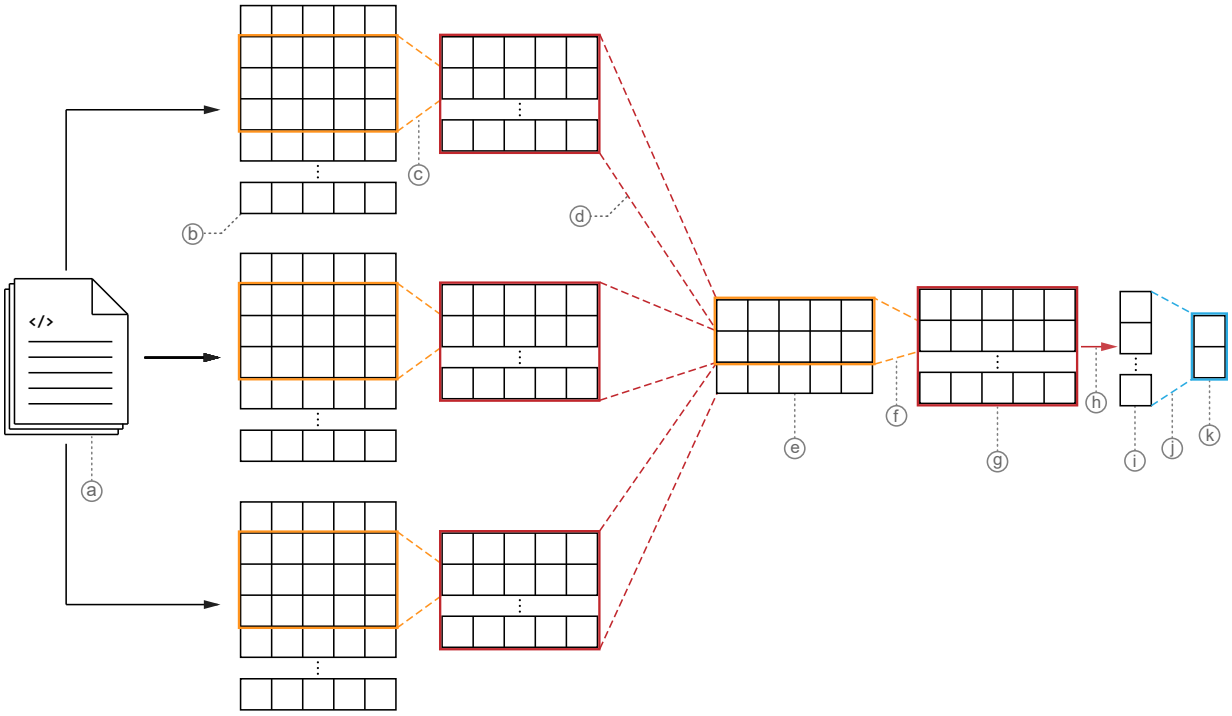


Figure 3.3: Illustration of H-CNN$_{\text{DIFF}}$, where the labels refer to the following: (a) source code diffs containing multiple files, (b) stacked token embeddings, (c) convolutional feature extraction, (d) max-pool across time, (e) file-level feature maps, (f) convolutional feature extraction, (g) stacked file-level feature vectors (h) max-pool across time, (i) commit-level feature vector, (j) fully connected softmax layer and (k) softmax output.

We also adapt the document classification baselines from Chapter 2 to learn on diff tokens: LSTM$_{\text{DIFF}}$ is a single-layer BiLSTM and R-LSTM$_{\text{DIFF}}$ is a regularized variant with weight dropping and embedding dropout.

3.4.3 Model

We design a simple hierarchical model (H-CNN$_{\text{DIFF}}$) that uses convolutional layers to progressively build a commit-level vector from token-level vectors, as illustrated in Figure 3.3. This high-level representation of the commit can then be used for classification. We represent each file in the commit diff as a concatenation of token-level vectors. These vector representations can be randomly initialized or obtained from pre-trained word embeddings. The input to our model is a concatenation of all file-level matrices in a commit diff.

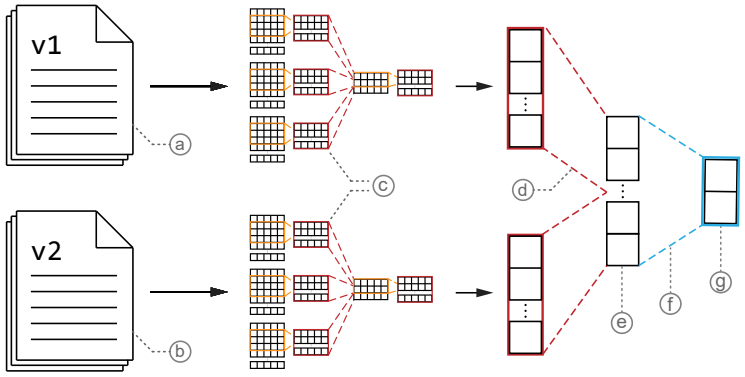


Figure 3.4: Illustration of H-CNN$_{\text{PAIR}}$, where the labels refer to the following: (a) source code containing multiple files before a commit, (b) source code containing multiple files after a commit, (c) H-CNN$_{\text{DIFF}}$ token and file encoders, (d) concatenation operation, (e) commit-level feature vector, (f) fully connected softmax layer and (g) softmax output.

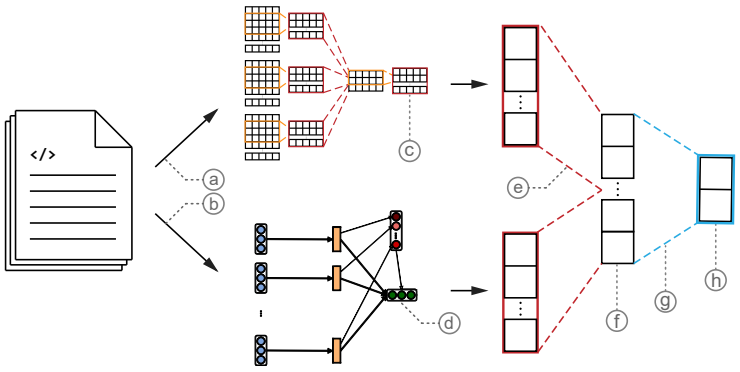


Figure 3.5: Illustration of Hybrid$_{\text{DIFF}}$, where the labels refer to the following: (a) diff tokens extracted from source code, (b) path-contexts extracted from source code, (c) H-CNN$_{\text{DIFF}}$ token and file encoders, (d) Code2Vec without softmax layer, (e) concatenation operation, (f) commit-level feature vector, (g) fully connected softmax layer and (h) softmax output.

Each file-level input matrix is passed through the token encoder that computes a file-level feature vector. The token encoder consists of three temporal convolutional layers in parallel, each having multiple filters of varying window sizes. A temporal max-pooling operation is applied to these feature maps to retain the feature with the highest value in every map.

We use a file encoder to compute a commit-level feature vector from these file-level feature vectors in a similar manner. This high-level commit representation is passed to a fully connected softmax layer that outputs the probability distribution over the security relevance of the commit.

We modify the architecture of our model for every research question according to the changes in the input representation.

For RQ2, we extract commit-level feature vectors before and after a commit with two separate sets of token and file encoders, as shown in Figure 3.4. We then concatenate the commit-level feature vectors and pass it through a fully connected softmax layer that outputs the required probability distribution. We require two sets of encoders as we maintain two separate vocabularies for the tokens extracted from Java classes before and after a commit, preventing us from adopting a siamese architecture.

For RQ3, we build a hybrid model that computes two commit-level feature vectors for each commit: the first from the token-based H-CNN$_{\text{DIFF}}$ and the second from the path-based Code2Vec. We concatenate these feature vectors and then obtain the required output probability distribution as discussed in RQ2. The architecture of this model is shown in Figure 3.5.

For RQ4, we incorporate the following regularization techniques into our model: DropBlock which was shown to be effective for object recognition [16], and embedding dropout which was shown to be effective for language modelling [28]. We apply embedding dropout on the inputs and DropBlock on the activations of the convolutional layers of H-CNN$_{\text{DIFF}}$ and H-CNN$_{\text{PAIR}}$; we refer to these variants as HR-CNN$_{\text{DIFF}}$ and HR-CNN$_{\text{PAIR}}$ respectively. We apply the same regularization techniques to the token-based component of Hybrid$_{\text{DIFF}}$ while retaining the original architecture of the path-based Code2Vec component; we refer to this variant as R-Hybrid$_{\text{DIFF}}$. As a regularized baseline, we also include R-LSTM$_{\text{DIFF}}$, an adapted version of R-LSTM$_{\text{TEXT}}$ from Chapter 2 that learns from diff tokens.

3.4.4 Training and Hyperparameters

We train all of our models for a maximum of 30 epochs with an early stopping criterion of 5 epochs. All the convolutional layers in the token encoder and the file encoder have three fil-

ter windows of size 3, 5, and 7 with 100 feature maps each. For HR-CNN$_{\text{DIFF}}$, HR-CNN$_{\text{PAIR}}$ and R-Hybrid$_{\text{DIFF}}$, we use embedding dropout and DropBlock with dropout rates of 0.2 and 0.1 respectively. We set a temporal block size of 4 for DropBlock. For LSTM$_{\text{DIFF}}$ and R-LSTM$_{\text{DIFF}}$, we choose a hidden state size of 128. In the case of R-LSTM$_{\text{DIFF}}$, we apply embedding dropout and weight dropping with dropout rates of 0.2 and 0.1 respectively.

For the path-based models, we sample a maximum of 200 path-contexts from all the files modified in a commit. Like Alon et al. [4], we did not notice any improvement in the results when increasing the number of contexts beyond 200.

For all the neural network models, we use Adam [21] for optimization, with a learning rate of 1.0×10^{-3} and batch size of 16. Further, we employ dropout with a rate of 0.5 on the fully connected layers of all neural network models for regularization.

3.5 Results and Discussion

We report the mean and standard deviation for accuracy and F_1 across fifty runs in Table 3.1. The best mean scores are bolded for each metric. While we also include the results of Commit2Vec (rows 5 and 6), it should be noted that Cabrera-Lozoya et al. [9] report results over a five-fold cross validation without separate validation and test sets. This difference could be exacerbated due to differences in implementation; Commit2Vec is built on Tensorflow, while our framework is based on PyTorch. Due to this difference in the experimental setup we have to be careful when making a comparison between any of the other models and Commit2Vec. The closest alternative is Code2Vec (rows 3 and 4), though it is not a replication of Commit2Vec since we pre-train the model differently as explained in Section 3.4.

We see that there is a considerable amount of overlap in the scores across different models in Table 3.1. While we can see that LR (row 1) and Code2Vec (rows 3 and 4) perform worse than rest of the bunch, it's hard to conclusively answer any of our research questions without analyzing these results further.

We use Tukey's test to find mean F_1 scores that are significantly different from each other. Tukey's test is a multiple comparison statistical test that simultaneously compares the means of every treatment to the means of every other treatment. The null hypothesis being tested is that all classifiers perform the same (i.e., have the same mean F_1 scores) and the observed differences are merely random [35]. In applying this test, we assume that our test F_1 scores are normally distributed and homoscedastic for each model. We apply this test individually for every research question on the test F_1 scores of the models from

Table 3.1: Results with mean and standard deviation across 50 runs on the test set

#	**Input Representation**	**Model**	**Embedding**	**Accuracy**	**F_1**
1	Bag of Words	LR	–	65.89	60.76
2		SVM	–	71.28	68.53
3	Bag of Paths	Code2Vec	Random	59.49 ±2.97	64.36 ±2.63
4			Pre-trained	57.86 ±2.40	60.55 ±2.61
5		Commit2Vec	Random	68.92 ±3.27	69.87 ±2.03
6			Pre-trained	72.12 ±0.82	71.83 ±0.66
7	Commit Diff Tokens	$\text{LSTM}_{\text{DIFF}}$	Random	69.70 ±4.44	73.41 ±2.48
8			Pre-trained	72.36 ±3.79	74.31 ±1.99
9		$\text{R-LSTM}_{\text{DIFF}}$	Random	70.73 ±4.81	74.21 ±3.00
10			Pre-trained	72.35 ±3.20	74.44 ±1.81
11		$\text{H-CNN}_{\text{DIFF}}$	Random	71.38 ±6.79	75.07 ±3.78
12			Pre-trained	67.49 ±5.66	72.00 ±3.07
13		$\text{HR-CNN}_{\text{DIFF}}$	Random	71.50 ±5.49	75.04 ±3.44
14			Pre-trained	66.15 ±6.53	71.26 ±3.15
15	Paired Souce Code Tokens	$\text{H-CNN}_{\text{PAIR}}$	Random	**74.48 ±3.45**	**76.63 ±2.54**
16			Pre-trained	70.62 ±3.37	73.09 ±2.41
17		$\text{HR-CNN}_{\text{PAIR}}$	Random	73.66 ±3.43	76.00 ±2.68
18			Pre-trained	69.32 ±4.91	72.16 ±2.63
19	Diff Tokens + Bag of Paths	$\text{Hybrid}_{\text{DIFF}}$	Random	70.70 ±5.67	72.33 ±4.05
20		$\text{R-Hybrid}_{\text{DIFF}}$	Random	68.53 ±5.46	71.26 ±4.32

Table 3.2: Results from Tukey's range test with a family-wise error rate of 0.05

#	Group 1		Group 2		Mean Diff.	P-adj	Reject?
	Model	Embedding	Model	Embedding			
			RQ1				
1	Code2Vec	Random	$\text{LSTM}_{\text{DIFF}}$	Random	9.05	0.001	Yes
2	Code2Vec	Pre-trained	$\text{LSTM}_{\text{DIFF}}$	Pre-trained	13.77	0.001	Yes
3	Code2Vec	Random	$\text{H-CNN}_{\text{DIFF}}$	Random	10.71	0.001	Yes
4	Code2Vec	Pre-trained	$\text{H-CNN}_{\text{DIFF}}$	Pre-trained	11.45	0.001	Yes
5	Code2Vec	Random	Code2Vec	Pre-trained	-3.82	0.001	Yes
6	$\text{H-CNN}_{\text{DIFF}}$	Random	$\text{H-CNN}_{\text{DIFF}}$	Pre-trained	-3.07	0.001	Yes
			RQ2				
7	$\text{H-CNN}_{\text{DIFF}}$	Random	$\text{H-CNN}_{\text{PAIR}}$	Random	1.56	0.048	Yes
8	$\text{H-CNN}_{\text{DIFF}}$	Pre-trained	$\text{H-CNN}_{\text{PAIR}}$	Pre-trained	1.09	0.265	No
9	$\text{H-CNN}_{\text{PAIR}}$	Random	$\text{H-CNN}_{\text{PAIR}}$	Pre-trained	-3.54	0.001	Yes
			RQ3				
10	Code2Vec	Random	$\text{Hybrid}_{\text{DIFF}}$	Random	7.97	0.001	Yes
11	$\text{H-CNN}_{\text{DIFF}}$	Random	$\text{Hybrid}_{\text{DIFF}}$	Random	-2.74	0.001	Yes
			RQ4				
12	$\text{LSTM}_{\text{DIFF}}$	Random	$\text{R-LSTM}_{\text{DIFF}}$	Random	0.80	0.744	No
13	$\text{H-CNN}_{\text{DIFF}}$	Random	$\text{HR-CNN}_{\text{DIFF}}$	Random	-0.03	0.900	No
14	$\text{H-CNN}_{\text{PAIR}}$	Random	$\text{HR-CNN}_{\text{PAIR}}$	Random	-0.63	0.900	No
15	$\text{Hybrid}_{\text{DIFF}}$	Random	$\text{R-Hybrid}_{\text{DIFF}}$	Random	-1.07	0.724	No

Table 3.1. We do not show all pairwise comparisons in the tables but only those pairs that are relevant to each research question.

RQ1: Are token-based models better than path-based models for identifying security-relevant commits?

Both randomly initialized and pre-trained $\text{LSTM}_{\text{DIFF}}$ models outperform their Code2Vec counterparts with a mean difference of around 9 and 14 points respectively (Table 3.2, rows 1 and 2); similarly both $\text{H-CNN}_{\text{DIFF}}$ models outperform Code2Vec with a mean difference of around 11 points (Table 3.2, rows 3 and 4). Interestingly, the non-neural SVM baseline (Table 3.1, row 2) has better accuracy and F_1 scores on average than Code2Vec, and the highest precision among all the models. This is a substantial improvement, especially considering that unlike Code2Vec or Commit2Vec, all of these models (Table 3.1, rows 1–2 and 7–14) are trained on input representations extracted from only the commit diff, without using the source code or the metadata present in a commit. With these results, we conclude that token-based models are better than path-based models for the identification of security-relevant commits.

It is worth noting that both Code2Vec and $\text{H-CNN}_{\text{DIFF}}$ perform worse when pre-trained than when initialized randomly (Table 3.2, rows 5 and 6). In part, this could be due to the embedding layer being frozen in the pre-trained model but not in the randomly initialized one. It could also be due to the relatively smaller sizes of the datasets we use for pre-training; we discuss this further in Section 3.6. We also see a similar pattern for Commit2Vec (Table 3.1, rows 5 and 6) when pre-trained on the semantic labelling task. One side-effect of pre-training embeddings is that it results in consistently less variation in the performance of these models across different runs as the random seed has no effect on the initialization of embeddings (Table 3.1, rows 7–18).

Based on the model's ability to predict method names in files across different projects, Alon et al. [4] claim that Code2Vec can be used for a wide range of programming language understanding tasks. However, for this specific task, most of our token-based models perform better than Code2Vec.

While Cabrera-Lozoya et al. [9] claim that pre-training tasks that are closer to the identification of security-relevant commits can improve the performance of Code2Vec, such a two-staged training process is a lot more involved as it requires a labelled training dataset and a relevant semi-supervised pre-training task along with the associated data. It might not always be possible to come up with a relevant task or a dataset large enough for pre-training.

Network architectures that are effective for a certain task, such as predicting method names, are not necessarily effective for other unrelated tasks. Choices between neural models should be made considering the nature of the task and the amount of training data available.

> Our results strongly show that token-based models are better than path-based models for the identification of security-relevant commits.

RQ2: Does extracting file-level features before and after a commit improve the identification of security-relevant commits?

Of all the models considered in this study, H-CNN$_{\text{PAIR}}$ achieves the highest mean accuracy and F_1 scores (Table 3.1, row 15). We see an improvement of around 1.5 points on average with a randomly-initialized H-CNN$_{\text{PAIR}}$ compared to its H-CNN$_{\text{DIFF}}$ counterpart (Table 3.2, row 7).

On the other hand, there is not enough evidence to show that pre-trained H-CNN$_{\text{PAIR}}$ performs better than pre-trained H-CNN$_{\text{DIFF}}$, even though the mean F_1 score has improved by more than one point (Table 3.2, row 8). In part, this could be due to the fact that H-CNN$_{\text{PAIR}}$ performs worse when pre-trained than when initialized randomly (Table 3.2, row 9), and the drop in performance is greater than what we see with H-CNN$_{\text{DIFF}}$.

While we see improvements due to the additional contextual information available in paired source code tokens, we feel that our dataset is not large enough to train these models to effectively use this additional information, and that we'd see larger gains with a bigger dataset.

> Providing our models with additional contextual information by training them on paired source code tokens does improve the identification of security-relevant commits.

RQ3: Do path-based and token-based models capture different signals when identifying security-relevant commits?

By combining the path-based Code2Vec and the token-based H-CNN$_{\text{DIFF}}$, we get a hybrid model that performs better than Code2Vec, but not H-CNN$_{\text{DIFF}}$ (3.2, rows 10 and 11). Despite the additional information that could be present in the bag of paths input, Hybrid$_{\text{DIFF}}$ achieves accuracy and F_1 in the same ballpark as our token-based models (3.1,

row 19). These results show that Code2Vec does not capture any additional information compared to H-CNN$_{\text{DIFF}}$ for this task.

Since using pre-trained embeddings did not result in a noticeable difference in performance for both Code2Vec and H-CNN$_{\text{DIFF}}$, we skip that variant for Hybrid$_{\text{DIFF}}$.

> While our results show that path-based models don't capture any additional information compared to token-based models for the identification of security relevant commits, it is not possible to confirm this without manual analysis of the predictions of these models and categorizing them based on the vulnerabilities in the dataset.

RQ4: Does regularization improve the effectiveness of our models in identifying security-relevant commits?

In the case of the R-LSTM$_{\text{DIFF}}$, where we apply embedding dropout and weight dropping, we see an increase in the mean F_1 score over LSTM$_{\text{DIFF}}$. However, this increase is lower than what we observed in Section 2.5 for document classification, and is not significant enough to reject the null hypothesis (Table 3.2, row 12). Likewise, we do not have sufficient evidence to show that applying embedding dropout and DropBlock affects the performance of HR-CNN$_{\text{DIFF}}$ and HR-CNN$_{\text{PAIR}}$ (Table 3.2, rows 13 and 14). In the latter scenario however, we actually see a drop in mean F_1 scores. It would be useful to conduct a comprehensive ablation study to determine the individual effect of these regularization techniques on these models across a range of SE tasks.

However, interpreting these results is less straightforward in the case of the R-Hybrid$_{\text{DIFF}}$ (Table 3.2, row 15). Regularization decreases the mean F_1 score, and to a greater degree than in the other cases. This might be due to the fact that we only apply regularization to the token-based H-CNN$_{\text{DIFF}}$ component, while retaining the original architecture of the path-based Code2Vec component. Since we primarily focus on token-based models, coming up with regularization techniques for path-based models is outside the scope of this study.

> We do not have sufficient evidence to show that applying embedding dropout, weight dropping or DropBlock for regularization improves the performance of our models.

Toward Robust Baselines. Looking at the standard deviation of the scores across different models allows us to compare the stability of these results. We feel that this is an important aspect to consider when training neural networks on smaller datasets, which is a typical scenario in SE. When training on a larger dataset, there is more data to order and

more weight updates to move away from a bad initialization [13]. However, on a smaller dataset like the one we use in this study, we can expect a much larger standard deviation of these scores across all models, as there are fewer weight updates to move away from a bad initialization. We see that this is indeed true, comparing the standard deviations from Table 3.1 to that of Table 2.2.

Most of the SE papers discussed in Section 3.2 that use neural networks in their approach treat scores as point estimates when comparing the performance of different models [4, 24, 41, 9]. While Cabrera-Lozoya et al. [9] do report standard deviations of their scores across five cross-validation splits, they draw their conclusions based only on a comparison of the mean scores.

3.6 Threats to Validity

The lexer and tokenizer we use from the javalang library target Java 8. We are not able to verify that all the projects and their forks in this study are using the same version of Java. However, we do not expect considerable differences in syntax between Java 7 and Java 8 except for the introduction of lambda expressions.[2]

There is also a question of to what extent the publicly disclosed vulnerabilities provided by Ponta et al. [36], used for evaluation in this study, represent the vulnerabilities found in real-world scenarios. While creating larger ground-truth datasets would always be helpful, it might not always be possible. To reduce the possibility of bias in our results, we ensure that we do not train commits from the same projects that we evaluate our models on. We also discard any commits belonging to the set of evaluation projects that are mined using regular expression matching.

Alon et al. [4] pre-train their model on 10M Java methods, much larger than the 700K Java methods we use in this study. It is possible that the performance of Code2Vec is considerably better than the results in Table 3.1 after pre-training on a larger dataset. Cabrera-Lozoya et al. [9] pre-train Commit2Vec on 12M Java methods and do not see an improvement compared to random initialization, thereby limiting this threat to validity, but not eliminating it due to differences in the experimental setup between our studies.

[2] https://www.jcp.org/en/jsr/detail?id=335

3.7 Conclusions

In this study, we evaluate five different input representations for the identification of security-relevant commits with a family of hierarchical neural network models. We show that models such as $\text{LSTM}_{\text{DIFF}}$ and $\text{H-CNN}_{\text{DIFF}}$ that learn on commit diff tokens are more effective than path-based models such as Code2Vec and Commit2Vec. In fact, we find no benefit in these pre-trained path-based models for this task. Not only is such a two-staged training process more time- and compute-intensive, but these models actually perform worse than a simple LSTM baseline. We also observe how $\text{H-CNN}_{\text{PAIR}}$ makes use of the additional contextual information available in paired source code tokens and improves upon $\text{H-CNN}_{\text{DIFF}}$. Finally, with the help of $\text{Hybrid}_{\text{DIFF}}$, we discuss why path-based models might not capture any additional information compared to token-based models for this task.

Chapter 4

Conclusions and Future Work

In this thesis, we show that simple neural models adapted from document classification, such as $\text{LSTM}_{\text{DIFF}}$ and $\text{H-CNN}_{\text{DIFF}}$, are effective in identifying security-relevant commits using just the commit diff tokens. In the process, we question the need for complex neural network architectures for document classification and establish strong neural baselines for the identification of security-relevant commits. As neural networks become more prevalent, one of the key takeaways for the SE community is the importance of robust baselines and an evaluation framework that compares performance metrics as distributions rather than individual data points. This would help researchers ensure the stability of their results and avoid erroneous conclusions [11].

While we only study the specific task of identifying security-relevant commits, our approach deserves exploration in other commit classification tasks as well. Such future research could be a step towards generalizing this thesis for commit classification altogether. Furthermore, it would be interesting to compare our models to the recent line of pre-trained Transformers, the state of the art for document classification.

Due to the limited number of ground-truth training samples, we are unable to exploit the full potential of deep learning. A reflection on the current state of labelled datasets in software engineering throws light on limited practicality of deep learning models for a lot of software engineering tasks [25]. This could potentially be solved with pre-trained deep language representation models for source code. Neural networks are becoming increasingly deeper and complex in the NLP literature, with significant interest in deep language representation models such as ELMo, GPT, and BERT [34, 38, 12]. Since all of these models are pre-trained in a semi-supervised manner, the vast amount of data available on GitHub can be used for this purpose. Such pre-trained models can be fine-tuned more

effectively on smaller datasets [19]. Ongoing research on Transformer-based contextual embeddings for source code is a step towards this goal [19, 14]. Drawing parallels with the recent history of NLP research, we are hoping that future research in this direction will considerably accelerate progress in tackling software problems with deep learning.

References

[1] Ashutosh Adhikari*, Achyudh Ram*, Raphael Tang, and Jimmy Lin. Rethinking complex neural network architectures for document classification. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4046–4051, 2019.

[2] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A convolutional attention network for extreme summarization of source code. In *33rd International Conference on Machine Learning*, pages 2091–2100, 2016.

[3] Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*, 2018.

[4] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: Learning distributed representations of code. *arXiv preprint arXiv:1803.09473*, 2018.

[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. A general path-based representation for predicting program properties. *ACM SIGPLAN Notices*, 53(4): 404–419, 2018.

[6] Chidanand Apté, Fred Damerau, and Sholom M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3): 233–251, 1994.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[8] Amiangshu Bosu, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. Identifying the characteristics of vulnerable code changes: An empirical study. In

Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 257–268. ACM, 2014.

[9] Rocío Cabrera-Lozoya, Arnaud Baumann, Antonino Sabetta, and Michele Bezzi. Commit2vec: Learning distributed representations of code changes. *arXiv preprint arXiv:1911.07605*, 2019.

[10] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 167–176, 2015.

[11] Matt Crane. Questionable answers in question answering research: reproducibility and variability of published results. *Transactions of the Association for Computational Linguistics*, 6:241–252, 2018.

[12] 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.

[13] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

[14] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[15] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027, 2016.

[16] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. DropBlock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10750–10760, 2018.

[17] Jordan Henkel, Shuvendu Lahiri, Ben Liblit, and Thomas Reps. Code vectors: Understanding programs through embedded abstracted symbolic traces. *arXiv preprint arXiv:1803.06686*, 2018.

[18] Hamel Husain. Towards natural language semantic code search, Sep 2018. URL https://githubengineering.com/.

[19] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. Pre-trained contextual embedding of source code. *arXiv preprint arXiv:2001.00059*, 2019.

[20] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[23] Rémi Lebret and Ronan Collobert. "The sum of its parts": Joint learning of word and phrase representations with autoencoders. *arXiv preprint arXiv:1506.05703*, 2015.

[24] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv preprint arXiv:1801.01681*, 2018.

[25] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. Sentiment analysis for software engineering: How far can we go? In *2018 IEEE/ACM 40th International Conference on Software Engineering*, pages 94–104. IEEE, 2018.

[26] Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship. *arXiv:1807.03341v2*, 2018.

[27] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, 2017.

[28] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[31] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating GitHub for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[35] Joaquın Pizarro, Elisa Guerrero, and Pedro L. Galindo. Multiple comparison procedures applied to model selection. *Neurocomputing*, 48(1-4):155–173, 2002.

[36] Serena E. Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. A manually-curated dataset of fixes to vulnerabilities of open-source software. *arXiv preprint arXiv:1902.02595*, 2019.

[37] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software. In *2018 IEEE International Conference on Software Maintenance and Evolution*, pages 449–460. IEEE, 2018.

[38] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[39] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: performance study of LSTM-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, 2017.

[40] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, 2017.

[41] Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 757–762. IEEE, 2018.

[42] Antonino Sabetta and Michele Bezzi. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International Conference on Software Maintenance and Evolution*, pages 579–582. IEEE, 2018.

[43] D. Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner's curse? On pace, progress, and empirical rigor. In *Proceedings of the 6th International Conference on Learning Representations, Workshop Track (ICLR 2018)*, 2018.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[45] Liran Tal. The state of open source security in 2019, Mar 2019. URL https://snyk.io/.

[46] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432, 2015.

[47] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[48] Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural Networks*, 71:1–10, 2015.

[49] Eran Yahav. From programs to interpretable deep models and back. In *International Conference on Computer Aided Verification*, pages 27–37. Springer, 2018.

[50] Pengcheng Yang, Xu Sun, Wei Li, Shuming Ma, Wei Wu, and Houfeng Wang. SGM: Sequence generation model for multi-label classification. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3915–3926, 2018.

[51] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[52] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

[53] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 253–263, 2017.

[54] Yaqin Zhou and Asankhaya Sharma. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 914–919. ACM, 2017.