# Co-Training for Commit Classification

**Jian Yi, David Lee**
DSO National Laboratories
12 Science Park Drive
Singapore 118225
ljianyi@dso.org.sg

**Hai Leong Chieu**
DSO National Laboratories
12 Science Park Drive
Singapore 118225
chaileon@dso.org.sg

## Abstract

Commits in version control systems (e.g. Git) track changes in a software project. Commits comprise noisy user-generated natural language and code patches. Automatic commit classification (CC) has been used to determine the type of code maintenance activities performed, as well as to detect bug fixes in code repositories. Much prior work occurs in the fully-supervised setting – a setting that can be a stretch in resource-scarce situations presenting difficulties in labeling commits. In this paper, we apply co-training, a semi-supervised learning method, to take advantage of the two views available – the commit message (natural language) and the code changes (programming language) – to improve commit classification.

## 1 Introduction

As a society's reliance on software grows, so will its needs for software analysis. Commits in version control systems (e.g. Git), a way of tracking changes made to code repositories, have been used in a range of analyses, including understanding maintenance activities (Hindle et al., 2009; Levin and Yehudai, 2017; Hönel et al., 2019), studying the impact of social factors in developer teams (Soto et al., 2017; Vasilescu et al., 2015), and detecting bug-fixing patches (Tian et al., 2012; Casalnuovo et al., 2017; Zafar et al., 2019).

Despite the usefulness of analyzing commits, there has been a lack of large-scale commit datasets. In 2017, to our knowledge, the largest public commit dataset (Levin and Yehudai, 2017) contained 1,151 commits categorized into three classes. For perspective, open source projects could see to tens of thousands of commits per year. For example, the Linux kernel's repository had 82,300 commits in 2019 (Foundation, 2021) and now comprises more than 1 million commits. The rate at which commits are created coupled with the variety of the kinds of

**Commit Message (NL view):**

```
Improve numerical stability of LayerNorm
    ↪ (#59987)
Summary:
Pull Request resolved: #59987
Similar as GroupNorm, improve numerical
    ↪ stability of LayerNorm by Welford
    ↪ algorithm and pairwise sum.
Test Plan: buck test mode/dev-nosan //caffe2/
    ↪ test:nn -- "LayerNorm"
Reviewed By: ngimel
Differential Revision: D29115235
fbshipit-source-id: 376
    ↪ dac89a4e14bd340aaaf169fef8d0d4ca4a1c4
```

**Code Change (PL view, cropped with ellipsis)**

```
diff --git a/aten/src/ATen/native/cpu/
    ↪ layer_norm_kernel.cpp b/aten/src/ATen/
    ↪ native/cpu/layer_norm_kernel.cpp
index 95a35571646d..366afe64b72a 100644
--- a/aten/src/ATen/native/cpu/
    ↪ layer_norm_kernel.cpp
+++ b/aten/src/ATen/native/cpu/
    ↪ layer_norm_kernel.cpp
@@ -1,13 +1,14 @@
 #include <cmath>
+#include <tuple>
...
 namespace at {
 namespace native {
@@ -29,30 +30,21 @@ void
    ↪ LayerNormKernelImplInternal(
   DCHECK_EQ(X.numel(), M * N);
   DCHECK(!gamma.defined() || gamma.numel()
    ↪ == N);
   DCHECK(!beta.defined() || beta.numel() ==
    ↪ N);
-  T* X_data = X.data_ptr<T>();
+  const T* X_data = X.data_ptr<T>();
...
```

Figure 1: Example of the NL-PL views of a commit.

commits likely pose difficulties in the creation of large-scale, quality commit datasets leading to data scarcity.

A commit consists of a commit message in natural language (NL) and code changes in programming languages (PL) (See Figure 1). Assuming weak dependence between the two views (NL and PL), a known sufficient condition for co-training to succeed (Abney, 2002), we apply iterative co-training (Blum and Mitchell, 1998) to train classifiers by bootstrapping from freely available unlabeled commit samples. Empirically, we show that co-training improves commit classification (CC) over a range of data scarcity scenarios.

With large transformer models achieving state-of-the-art performance in NLP tasks, Feng et al. (2020) created CodeBERT, a pretrained bimodal model based on RoBERTa's (Liu et al., 2019) architecture (another large language model) for natural language and code to capture the semantic connection between both modalities via an attention mechanism to output representations that can support NL-PL understanding tasks. Due to the presence of NL and PL views in commits, we propose fine-tuning a pretrained CodeBERT for transfer learning on the CC task.

In this paper we examine the following questions: **RQ1:** Can attention between NL and PL improve classification? **RQ2:** Is CodeBERT better than RoBERTa at representing commit messages for CC? **RQ3:** Can co-training improve CC? While RQ3 is our main contribution, we study RQ1 and RQ2 to decide on the neural network architecture.

## 2 Related Work

**Commit Classification (CC)** Over the years, researchers have applied a variety of machine learning methods to CC. Hindle et al. (2009) applied decision trees, naive Bayes, SVM (Support Vector Machine), and nearest-neighbor algorithms to classify commits into maintenance categories. Tian et al. (2012) used a SVM on engineered features from commit messages (as a bag of words) and code changes (parsed for the number of code constructs) to classify bug-fixing patches in the Linux kernel, remarking that bug fixes typically involve code changes at a single location while non-bug-fixing commits involve more lines of code. Levin and Yehudai (2017) used word frequencies in commit messages and code changes as inputs to decision trees, GBM (Gradient Boosting Machine), and RF (Random Forest) to classify commits into maintenance activities. Zhou and Sharma (2017) applied logistic regression on the outputs of RF, Gaussian Naive Bayes, K-Nearest Neighbours, GBM, and Adaboost to classify vulnerability-related commits, remarking that commit messages, particularly short ones, may not be unique enough to distinguish vulnerability-relatedness. Sabetta and Bezzi (2018) used a SVM on bags of words obtained from commit messages and another SVM on bags of words from code changes to classify security-relevant commits, remarking that code changes contain names (e.g., variables, functions) that convey semantics and could be treated as a text document. Hönel et al. (2019) used a variety of methods on the code density of commits to classify commits into maintenance activities. Zafar et al. (2019) fine-tuned a pre-trained BERT model (Devlin et al., 2018) with a single neuron perceptron head to classify bug-fixing commits. Keshav Ram (2020) used a linear SVM on TF-IDF features of Java tokens extracted from code changes, Code2Vec (Alon et al., 2019), convolutional neural network on code changes with surrounding code as context, and BiLSTM (Bi-directional Long Short-Term Memory) on code changes without context to classify commits, remarking that the Code2Vec approach yielded poor results. (Lozoya et al., 2021) proposed a method inspired by Code2Vec to compute vectors from code changes, trained on the pretext task of classifying commits' Jira Ticket Priorities before classifying security-fix commits. Many aforementioned works were conducted under the fully-supervised setting — a setting that can be a stretch for new categories where labeled data is scarce.

With large language models achieving state-of-the-art performance in NLP tasks, Feng et al. (2020) created CodeBERT, a pre-trained bimodal model based on RoBERTa's (Liu et al., 2019) architecture (another large language model) for natural language and code in an attempt to capture the semantic connection between both modalities to output representations that can support NL-PL understanding tasks. CodeBERT is pre-trained on code-documentation pairs from Github repositories with the masked language modeling objective (used in pre-training BERT), as well as the replaced token detection objective (used in pre-training ELECTRA (Clark et al., 2020)). CodeBERT is bi-modal in the sense that natural language and code can be concatenated into a single sequence and fed into the model to give rise to attention between code and natural language tokens. In this paper, we adopt a fine-tuning approach to adapt RoBERTa and CodeBERT models for CC.

**Datasets** Ghadhab et al. (2021) combined three datasets (Levin and Yehudai, 2017; Mauczka et al., 2015; AlOmar et al., 2019) containing commits from open-source projects that cover a range of domains such as databases, programming languages and integration frameworks to create a dataset with 1,793 labeled commits for classifying types of software maintenance activities. Another data set was compiled by Ponta et al. (2019), providing 1,282 commits that fix vulnerabilities in 205 open-source Java projects for classifying security commits.

To our knowledge, the largest data set (that we denote as **RA-Data**) was compiled by Reis and Abreu (2021), aggregating three datasets (Ponta et al., 2019; Reis and Abreu, 2017; Fan et al., 2020) as well as additional scraping to provide 8,057 security-relevant commits and about 110,000 non-security-relevant commits spanning 1,339 open-source projects and 20 programming languages. In this paper, we use RA-Data as it is larger and more varied than the other datasets. To our knowledge, our work is the first to attempt CC on this dataset.

**Co-Training** Co-training was first proposed by Blum and Mitchell (1998) and applied to classify university web pages by assuming that the web pages' text and inbound links are two distinct views conditionally independent to the other given the class label, and sufficient for learning by itself. During each co-training iteration, the classifiers' most confident predictions for each view are added to an initially small set of labeled training data, expanding it. In the same iteration, the classifiers are further trained on the expanded dataset. Later, Abney (2002) proved that weak dependence between views, a weaker assumption than conditional independence, is sufficient for co-training. Next, Balcan et al. (2005) proved an even weaker sufficient condition they term expansion assumption. After that, Wang and Zhou (2010) showed that combinative label propagation over two views is both sufficient and necessary for co-training.

Ling et al. (2009) empirically verifies on 32 datasets that improvements by co-training are likelier the more independent the two views are and the more sufficient each view is, publishing results supporting aforementioned sufficient conditions for co-training.

In this work, we observed that commits possess a natural view split – commit message (natural language) and code changes (programming language) – that one could reasonably hope for the weak dependence assumption as well as the sufficiency for learning from one view assumption to hold. If true, co-training is a natural fit for learning on commit datasets.

## 3 The 900Repo Dataset

To create a balanced dataset from RA-Data, we first noted that about 910 repositories overlap in both classes – security ("positive" class) & non-security ("negative" class). The remaining repositories occur in one class but not the other. We extracted all 3,765 positive samples belonging to these 910 overlapping repositories and randomly sampled twice as many negative samples from the same 910 repositories, giving a total of 10,000 commits. As at August 2021, due to RA-Data not having code changes (a.k.a. diffs), we scraped for the code changes via Github's API. The result is a dataset with 3,765 positive samples and roughly 6,300 negative samples that we refer to as **900Repo**. This dataset is publicly available at `https://github.com/davidleejy/wnut21-cotrain`.

## 4 Pre-trained models

**RQ1.** While it seems beneficial to pack the NL and PL views into a single input sequence to a transformer model so that attention between message tokens and code tokens could improve the representation, such a sequence is likely to exceed the maximum input sequence length even in large models like RoBERTa & CodeBERT. As such, striving for attention between NL and PL views often means truncating both modalities significantly as a trade-off, losing information in the process. An alternative is to feed each modality into a separate transformer model, effectively sacrificing attention between modalities to gain information (fewer tokens truncated), and attaching, for example, a single MLP (Multi-Layer Perceptron) at both heads of these transformer models to yield a joint representation upon which classification can be performed. Our experiments in Section 4.1 suggest that sacrificing attention to allow longer text inputs is the better approach.

**RQ2.** As CodeBERT was pre-trained with the aim of capturing the semantic connection between NL and PL, one might expect CodeBERT to produce better representations for commit messages than the architecturally-identical RoBERTa which was not trained with NL-PL understanding in mind. Our experiments in Section 4.1 suggest that the representations generated by RoBERTa and CodeBERT for commit messages offer similar performance outcomes for CC.

### 4.1 Models and Experiments

The 900Repo dataset is split with a 70:15:15 train:validation:test ratio. We fine-tune the MLP to the CC task (binary classification) via gradi-

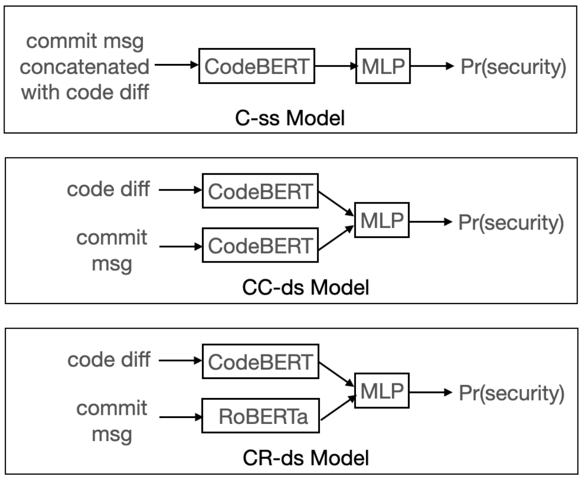

Figure 2: Neural network architectures considered.

ent descent with ADAM. Pre-trained CodeBERT and RoBERTa weights are not modified, each with 125M parameters. We explore three architectural configurations (See Figure 2) to investigate RQ1 and RQ2:

**C-ss** (CodeBERT, single sequence): Both modalities, NL and PL, are concatenated and truncated into a single 512 token sequence (hence "single sequence"), before being fed into a pre-trained CodeBERT model. Commit messages are roughly given 30% of the 512 token sequence limit and code changes are given the remaining 70%. An MLP is attached to the head of the CodeBERT model for fine-tuning. The classification prediction is output by the MLP.

**CC-ds** (CodeBERT & CodeBERT, distinct sequences): Both modalities are fed into separate, identical pre-trained CodeBERT models (hence "distinct sequences"). As such, there is no attention across modalities. Both modalities are allowed the maximum 512 token sequence limit and have its own embedding. A single MLP is attached to both heads of the two separate CodeBERT models (one for NL, one for PL) to yield a joint representation. The classification prediction is output by the MLP.

**CR-ds** (CodeBERT & RoBERTa, distinct sequences): Identical to CC-ds except the NL view is fed to RoBERTa instead of CodeBERT.

**Results:** Table 1 shows the performance of the three different architectural configurations. Both ds (distinct sequences) models perform better than the C-ss (single sequence) model, suggesting that sacrificing attention between the modalities in return for

| Model | P | R | F1 | AUC | Acc. |
|---|---|---|---|---|---|
| C-ss | 80.3 | 80.4 | 80.4 | 81.0 | 80.4 |
| CC-ds | 83.0 | 83.0 | 83.0 | 84.9 | 83.0 |
| CR-ds | **84.0** | **84.1** | **83.9** | **87.4** | **84.1** |

Table 1: Test results with different architectures. P/R/F1: average weighted precision, recall and F1 (weighted by class-size), Acc.: accuracy, AUC: area under precision-recall curve.

```
Algorithm 1: Co-training
  Note: Denote positive class as ⊞; negative
        class as ⊠.
  Input: Confidence $c$; Confidence limit $C$;
         Min. (resp. max.) of samples to
         label as ⊞ (resp. ⊠):
         $n_{min}(⊞), n_{max}(⊞)$ (resp.
         $n_{min}(⊠), n_{max}(⊠)$)
  Data: Labeled $T$, Unlabeled $U$, Validation
        $V$.
  Output: Trained classifiers $NL$, $PL$.
1 Train on $T$ to obtain $NL$ and $PL$ models
2 while $U$ not empty do
3     for $M$ in $NL$, $PL$ do
4         Label $U$ with $M$
5         for $L$ in ⊞, ⊠ do
6             Select samples with predicted
              probability $> 1 - c$ when $L$ is
              ⊞ (resp. $< c$ when $L$ is ⊠).
              Denote selection's size as $n_L$.
7             If $n_L > n_{max}(L)$ limit, truncate
              selection to size $n_{max}(L)$.
8             If $n_L < n_{min}(L)$ limit, pad
              selection to size $n_{min}(L)$.
9     Add selected samples to $T$.
10    Train new models $NL_{new}$ and $PL_{new}$
      on T. If performance on $V$ improves,
      set $NL = NL_{new}$ and $PL = PL_{new}$.
11    Increment $c$. Maintain $c < C$.
```

being able to pass more information into the model is a reasonable trade-off for CC. The CR-ds (CodeBERT & RoBERTa) model slightly outperformed the CC-ds (CodeBERT & CodeBERT) model suggesting that CodeBERT does not offer an advantage over RoBERTa at representing commit messages for the CC task.

| T | Model | P | R | F1 | Acc. |
|---|---|---|---|---|---|
| 0.5% | NL-T | 67.8 | 68.8 | **66.7** | 68.8 |
| | PL-T | 61.2 | 63.0 | 61.4 | 63.0 |
| | NL | **68.0** | **69.0** | **66.7** | **69.0** |
| | PL | 64.7 | 66.3 | 63.4 | 66.3 |
| 1% | NL-T | 58.5 | 62.3 | 57.2 | 62.3 |
| | PL-T | 65.2 | 65.3 | 65.3 | 65.3 |
| | NL | **73.0** | **72.2** | **72.5** | **72.2** |
| | PL | 68.5 | 67.3 | 60.9 | 67.3 |
| 2% | NL-T | 75.9 | 76.3 | 76.0 | 76.3 |
| | PL-T | 68.4 | 65.9 | 66.5 | 65.9 |
| | NL | **76.2** | **76.5** | **76.3** | **76.5** |
| | PL | 72.4 | 73.1 | 72.4 | 73.1 |
| 4% | NL-T | 77.6 | 77.0 | 76.9 | 77.0 |
| | PL-T | 71.2 | 69.3 | 69.7 | 69.3 |
| | NL | **77.7** | **77.9** | **77.8** | **77.9** |
| | PL | 73.4 | 70.6 | 70.9 | 70.6 |

Table 2: Co-training experimental results. T: labeled training data as a percentage of the dataset. NL-T, PL-T: supervised models trained only on T. NL, PL: co-trained models. P/R/F1: average weighted precision, recall, and F1 score. Acc.: accuracy.

## 5 Co-training for commit classification

In the previous section, a joint representation of the two modalities is obtained by attaching a single MLP to the outputs of the transformer models. However, here the CodeBERT & RoBERTa outputs are directed to separate MLP's. For convenience, we denote as **NL** the RoBERTa-MLP classifier model on commit messages, and as **PL** the CodeBERT-MLP classifier model on code changes.

To simulate resource-poor conditions, we use either 0.5%, 1%, 2% or 4% of the 900Repo dataset as labeled training data with a validation set of the same size, and 15% of the data as test set. The remaining is used as unlabeled data for co-training. The validation data is used by supervised baselines for model selection (i.e. selecting the epoch during training with best performance on validation set).

We show our co-training algorithm in Algorithm 1. In each iteration, a small amount of confidently labeled commits are added to the training dataset which in turn is used in the next iteration's training. At all times, only the MLP portions are fine-tuned via gradient descent with ADAM for CC. Pre-trained CodeBERT and RoBERTa weights are frozen. In all our experiments, we set $c = 0.001$, $C = 0.15$, $n_{max}(\boxplus) = 3$, $n_{max}(\boxtimes) = 5$, $n_{min}(\boxplus) = n_{min}(\boxtimes) = 1$.

**Results:** Table 2 shows the test performance for resource-poor scenarios ($T = 4\%$, $2\%$, $1\%$, & $0.5\%$ labeled data). Co-training provided an improvement across virtually all performance metrics in all scenarios. In all scenarios, the PL classifier (CodeBERT-MLP) improved more than the NL classifier (RoBERTa-MLP).

Under both supervised & co-training approaches, the NL classifier is observed to perform better than the PL classifier. This could be due to the challenging nature of obtaining good representations for programming languages as well as the length of code changes frequently being much longer than the commit message and CodeBERT's 512 token limit.

## 6 Conclusion

We found co-training to be helpful for CC in resource-poor settings where there are insufficient labeled commits for supervised approaches. Further research directions could explore the use of semi-supervised learning methods to expand the capabilities of semantic search within code repositories. Another limitation of our current work is that commits are often too long for Roberta and CodeBert. For future work, we could apply models that allow longer sequences as input, e.g., the Longformer (Beltagy et al., 2020).

## References

Steven Abney. 2002. Bootstrapping. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 360–367.

Eman AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni. 2019. Can refactoring be self-affirmed? an exploratory study on how developers document their refactoring activities in commit messages. In *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*, pages 51–58. IEEE.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29.

Maria-Florina Balcan, Avrim Blum, and Ke Yang. 2005. Co-training and expansion: Towards bridging theory and practice. *Advances in neural information processing systems*, 17:89–96.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100.

Casey Casalnuovo, Yagnik Suchak, Baishakhi Ray, and Cindy Rubio-González. 2017. Gitcproc: A tool for processing and classifying github commits. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 396–399.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. Ac/c++ code vulnerability dataset with code changes and cve summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 508–512.

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

The Linux Foundation. 2021. 2020 linux kernel history report. -.

Lobna Ghadhab, Ilyes Jenhani, Mohamed Wiem Mkaouer, and Montassar Ben Messaoud. 2021. Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. *Information and Software Technology*, 135:106566.

Abram Hindle, Daniel M German, Michael W Godfrey, and Richard C Holt. 2009. Automatic classication of large changes into maintenance categories. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 30–39. IEEE.

Sebastian Hönel, Morgan Ericsson, Welf Löwe, and Anna Wingkvist. 2019. Importance and aptitude of source code density for commit classification into maintenance activities. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 109–120. IEEE.

Achyudh Ram Keshav Ram. 2020. Exploiting token and path-based representations of code for identifying security-relevant commits. Master's thesis, University of Waterloo.

Stanislav Levin and Amiram Yehudai. 2017. Boosting automatic commit classification into maintenance activities by utilizing source code changes. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 97–106.

Charles X Ling, Jun Du, and Zhi-Hua Zhou. 2009. When does co-training work in real data? In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 596–603. Springer.

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 preprint arXiv:1907.11692*.

Rocío Cabrera Lozoya, Arnaud Baumann, Antonino Sabetta, and Michele Bezzi. 2021. Commit2vec: Learning distributed representations of code changes. *SN Computer Science*, 2(3):1–16.

Andreas Mauczka, Florian Brosch, Christian Schanes, and Thomas Grechenig. 2015. Dataset of developer-labeled commit messages. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 490–493. IEEE.

Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. 2019. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 383–387. IEEE.

Sofia Reis and Rui Abreu. 2017. Secbench: A database of real security vulnerabilities. In *SecSE@ ESORICS*, pages 69–85.

Sofia Oliveira Reis and Rui Abreu. 2021. A ground-truth dataset of real security patches. *arXiv*.

Antonino Sabetta and Michele Bezzi. 2018. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International conference on software maintenance and evolution (ICSME)*, pages 579–582. IEEE.

Mauricio Soto, Zack Coker, and Claire Le Goues. 2017. Analyzing the impact of social attributes on commit integration success. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 483–486. IEEE.

Yuan Tian, Julia Lawall, and David Lo. 2012. Identifying linux bug fixing patches. In *2012 34th international conference on software engineering (ICSE)*, pages 386–396. IEEE.

Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. 2015. A data set for social diversity studies of github teams. In *2015 IEEE/ACM 12th working conference on mining software repositories*, pages 514–517. IEEE.

Wei Wang and Zhi-Hua Zhou. 2010. A new analysis of co-training. In *ICML*.

Sarim Zafar, Muhammad Zubair Malik, and Gursimran Singh Walia. 2019. Towards standardizing and improving classification of bug-fix commits. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE.

Yaqin Zhou and Asankhaya Sharma. 2017. 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.