STRATA: BUILDING ROBUSTNESS WITH A SIMPLE METHOD FOR GENERATING BLACK-BOX ADVERSARIAL ATTACKS FOR MODELS OF CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

Adversarial examples are imperceptible perturbations in the input to a neural model that result in misclassification. Generating adversarial examples for source code poses an additional challenge compared to the domains of images and natural language, because source code perturbations must adhere to strict semantic guidelines so the resulting programs retain the functional meaning of the code. We propose a simple and efficient black-box method for generating state-of-the-art adversarial examples on models of code. Our method generates untargeted and targeted attacks, and empirically outperforms competing gradient-based methods with less information and less computational effort. We also use adversarial training to construct a model robust to these attacks; our attack reduces the F1 score of CODE2SEQ by 42%. Adversarial training brings the F1 score on adversarial examples up to 99% of baseline.

1 INTRODUCTION

Although machine learning has been shown to be effective at a wide variety of tasks across computing, statistical models are susceptible to *adversarial examples*. Adversarial examples, first identified in the continuous domain by Szegedy et al. (2014), are imperceptible perturbations to input that result in misclassification. Researchers have developed effective techniques for adversarial example generation in the image domain (Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2017; Papernot et al., 2016a) and in the natural language domain (Alzantot et al., 2018; Belinkov & Bisk, 2018; Cheng et al., 2020; Ebrahimi et al., 2018; Michel et al., 2019; Papernot et al., 2016b), although work in the source code domain is less extensive (see Related Work). The development of adversarial examples for deep learning models has progressed in tandem with the development of methods to make models which are robust to such adversarial attacks, though much is still being learned about model robustness (Goodfellow et al., 2015; Madry et al., 2018; Shafahi et al., 2019; Wong et al., 2019).

The threat of adversarial examples poses severe risks for ML-based malware defenses (Al-Dujaili et al., 2018; Grosse et al., 2016; Kaur & Kaur, 2015; Kolosnjaji et al., 2018; Kreuk et al., 2019; Suciu et al., 2019), and introduces the ability of malicious actors to trick ML-based code-suggestion tools to suggest bugs to an unknowing developer (Schuster et al., 2020). Generating adversarial examples for models of code poses a challenge compared to the image and natural language domain, since the input data is discrete and textual and adversarial perturbations must abide by strict syntactical rules and semantic requirements. The CODE2SEQ model is a state-of-the-art model of code that has been used to explore adversarial example design and robustness methods on models of code (Rabin & Alipour, 2020; Ramakrishnan et al., 2020).

In this work, we propose the Simple TRAined Token Attack (STRATA), a novel and effective method for generating black-box and white-box adversarial attacks against CODE2SEQ. Our method replaces local variable names with high impact candidates that are identified by dataset statistics. It can also be used effectively for targeted attacks, where the perturbation targets a specific (altered) output classification. Further, we demonstrate that adversarial training, that is, injecting adversarial examples into CODE2SEQ's training set, improves the robustness of CODE2SEQ to adversarial attacks.

We evaluate STRATA on CODE2SEQ, though we hypothesize that the method can be applied to other models. The principles underlying STRATA apply not only to models of source code, but also to natural language models in contexts where the vocabulary is large and there is limited training data.

STRATA has a number of advantages compared to previously proposed adversarial attack strategies:

1. STRATA constructs state-of-the-art adversarial examples that outperform gradient-based methods;
2. STRATA utilizes white-box or black-box adversarial example generation, with black-box attack performance close to that of the white-box;
3. STRATA does not require the use of a GPU and can be executed more quickly than competing gradient-based attacks (however, we leave the discussion of this result to the Appendix);
4. STRATA is the only available method (known to the authors at present) which performs targeted attacks on CODE2SEQ.

2 MOTIVATION

CODE2SEQ, developed by Alon et al. (2019a), is an encoder-decoder model inspired by SEQ2SEQ (Sutskever et al., 2014); it operates on code rather than natural language. The model is tasked to predict method names from the source code body of a Java method, using a corpus of approximately 16 million Java methods from top-starred public GitHub repositories for training data. The model considers both the structure of an input program's Abstract Syntax Trees (ASTs) as well as the tokens corresponding to identifiers such as variable names, types, and invoked method names. To reduce the vocabulary size, identifier tokens are split into subtokens by commonly used delimiters such as camelCase and under_scores. In this example, subtokens would include "camel" and "case" and "under" and "scores". CODE2SEQ encodes subtokens into distributed embedding vectors. These subtoken embedding vectors are trained to capture semantic structure, so nearby embedding vectors should correspond to semantically similar subtokens (Bengio et al., 2003). In this paper, we distinguish between *subtoken* embedding vectors and *token* embedding vectors. *Subtoken* embedding vectors are trained model parameters. *Token* embedding vectors are computed as a sum of the embedding vectors of the constituent subtokens. If the token contains more than five subtokens, only the first five are summed, as per the CODE2SEQ architecture. The full description and architecture of the CODE2SEQ model is given in the original paper by Alon et al. (2019a).

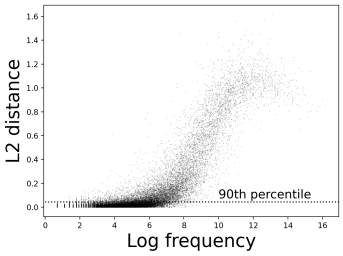
The CODE2SEQ model only updates a subtoken embedding as frequently as that subtoken appears during training, which is proportional to its representation in the training dataset. However, the training datasets have very large vocabularies consisting not only of standard programming language keywords, but also a huge quantity of neologisms. The frequency at which subtokens appear in the CODE2SEQ java-large training set varies over many orders of magnitude, with the least common subtokens appearing fewer than 150 times, and the most common over 10^8 times.

Thus, subtoken embedding vectors corresponding with infrequently-appearing subtokens will be modified by the training procedure much less often than common subtokens. Figure 1a demonstrates this phenomenon, showing a disparity between L2 norms of frequent and infrequently-appearing subtoken embedding vectors.

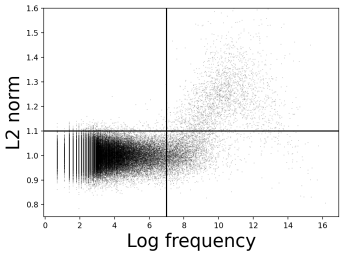
We confirm this empirically. When we initialized embedding vectors uniformly at random and then trained the model as normal, as per Alon et al. (2019a), we found that the vast majority of final, i.e., post-training, embedding vectors change very little from their initialization value. In fact, 90% of embedding tokens had an L2 distance of less than 0.05 between the initial vector and final, post-training vector. About 10% of subtokens had a large L2 distance between the initial embedding and final embedding; these subtokens were more frequent in the training dataset and had embedding vectors with a notably larger final L2 magnitude (Figure 1).

The observation that high-L2-norm embedding vectors are associated with subtokens that appear sufficiently frequently in the dataset motivates the core intuitions of our attack[1]. We show in this

[1]We note very-high-frequency subtokens have small L2 norms. Examples of these very-high-frequency subtokens include: `get`, `set`, `string`, and `void`, which appear so often as to not be useful for classification. Despite the fact that these subtokens are not good adversarial candidates for STRATA, there are so few of them that we expect them to have minimal influence on the effectiveness of our attack.



(a) The L2 distance between pre- and post-training embedding vectors for each subtoken, plotted against the log frequency of subtokens in the training dataset.



(b) L2 norm of trained subtoken embedding vectors. Dashed and dotted lines chosen arbitrarily to illustrate tokens which have high L2 norm or frequency.

Figure 1: Visualization of embedding vectors for CODE2SEQ-SM.

paper that subtokens with high-L2-norm embedding vectors can be used for effective adversarial examples, which are constructed as follows:

1. To maximize adversarial effectiveness in a white-box setting, we should use tokens with high L2 norm embedding vectors as local variable name replacements. We confirm this empirically in the Experiments section.
2. In the absence of information about the L2 norms of embedding vectors, we can isolate high-L2-norm subtokens for local variable name replacement by selecting tokens which appear in the training dataset often enough to be well trained. This is empirically confirmed by the large intersection of high-L2-norm subtokens and subtokens with high frequency.

3 METHODS

3.1 DATASET DEFINITIONS AND CONSIDERATIONS

Alon et al. (2019a) presents three datasets that were used for training different CODE2SEQ models: java-small (700k examples), java-medium (4M examples), and java-large (16M examples). Unsurprisingly, CODE2SEQ performs best on java-large (Alon et al., 2019a). We also used these three datasets to train three different models: we disambiguate the trained CODE2SEQ models from the datasets by denoting them CODE2SEQ-SM, CODE2SEQ-MD, CODE2SEQ-LG for models trained on java-small, -medium, and -large, respectively. Many of our experiments are evaluated on all three models; however, experiments that require adversarial training are only evaluated on CODE2SEQ-SM, for computational feasibility.

3.2 THE ATTACK

Traditional adversarial attacks on discrete spaces involve searching the discrete space for semantically similar perturbations that yield a misclassification. Searching the space of all possible valid discrete changes in source code is often intractable or even impossible (Rice, 1953). However, there are strategies to reduce the search space. For example, perturbations may be limited to a small number of predefined operations which are known to be semantically equivalent, such as inserting deadcode, replacing local variable names, or replacing expressions with known equivalent expressions. Gradient-based attacks on the embedding space may also be used in order to optimize the search of the space itself (Ramakrishnan et al., 2020; Yefet et al., 2020). Gradient-based attacks rely on knowledge of the exact parameters of the model.

We propose STRATA, which replaces local variable names with high-impact subtokens to generate adversarial examples. STRATA leverages the fact that only a relatively small number of subtoken embeddings are critical for the classification task performed by the CODE2SEQ model.

In Section 2, we presented two ways to identify high-impact subtokens. STRATA will use these to replace local variable names. Recall that the model composes these subtokens into tokens by summing the first five constituent subtoken embedding vectors. We wish to maximize the L2 norm of the resulting token, while minimizing the semantic change. We propose three strategies:

1. *single:* pick a single subtoken as the token;
2. *5-diff:* pick five different (not necessarily unique) subtokens and concatenate them, which will have a higher expected L2 norm than *single*;
3. *5-same:* pick a single subtoken, and repeat the subtoken five times to form a token, which will have the largest expected L2 norm, by the triangle inequality[2].

We subjectively propose that *single* is the smallest semantic change, *5-same* is the largest change, and *5-diff* represents a medium.

For a given Java method, STRATA generates an untargeted perturbation as follows:

1. Select one random local variable v;
2. Choose an adversarial token v^* appropriately, using the chosen concatenation strategy (*single*, *5-diff*, or *5-same*). For white-box attacks, choose each subtoken from a high-L2-norm vocabulary (top-n by L2 norm). For black-box attacks, choose each subtoken with sufficiently high frequency (top-n by frequency). We discuss the optimal cutoff values (n) for L2 and frequency in Section 3.5.
3. Replace v with v^*.

Similarly, to perform targeted attacks in which we want the output to include a particular subtoken t, we perform the same steps as the untargeted attack, and choose v^* to be a *5-same* concatenation of t.

3.3 DATASET-AGNOSTIC BLACK-BOX ATTACKS

The black-box version of STRATA, as proposed, utilizes the subtoken distribution in the training dataset of the targeted model to determine high-impact subtoken candidates. This technique thus relies on knowledge of the original dataset, which may not always be known or available. Nonetheless, STRATA can generate effective adversarial attacks even without the training dataset. Rather, we can determine subtoken frequency statistics from a different (potentially non-overlapping) Java dataset. We empirically confirm that STRATA can use non-overlapping datasets (java-small, java-medium, and java-large) in this way to attack CODE2SEQ models that have been trained on a different dataset. We conclude that the subtoken distributions of non-overlapping Java datasets are sufficiently similar for STRATA to be effective. This confirms that STRATA can be applied in a completely black-box setting without knowledge of the model parameters or training dataset. We leave the results of this experiment to the appendix.

3.4 ADVERSARIAL TRAINING

We perform standard adversarial training in order to make CODE2SEQ most robust to adversarial attacks. To test the robustness of an adversarially-trained CODE2SEQ model, we perform the following experiment:

1. We train a CODE2SEQ-SM model.
2. We generate STRATA perturbations of the java-small training dataset using a vocabulary of the top-1000 subtokens by L2 norm of the embedding. For each training example, we generate five adversarial perturbations for each concatenation strategy, *single*, *5-diff*, and *5-same*.
3. We fine-tune the CODE2SEQ-SM model with the adversarial examples for a final epoch.

3.5 IMPLEMENTATION

We assayed the effectiveness of the method by applying a simple local variable token replacement attack. We select a single locally-declared variable at random and rename it with an adversarial token which does not conflict with another name. Because the initial variable is locally declared, we know that changing the name will have no effect elsewhere in the program, and will have no behavioral effect. The token replacement, however, can effectively attack CODE2SEQ; example shown in the Appendix.

[2]The triangle inequality states that $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$, and is equal (and thus maximized) when $\mathbf{x}$ and $\mathbf{y}$ are colinear, which occurs when $\mathbf{x} = \mathbf{y}$. This is easily generalized to five vectors.

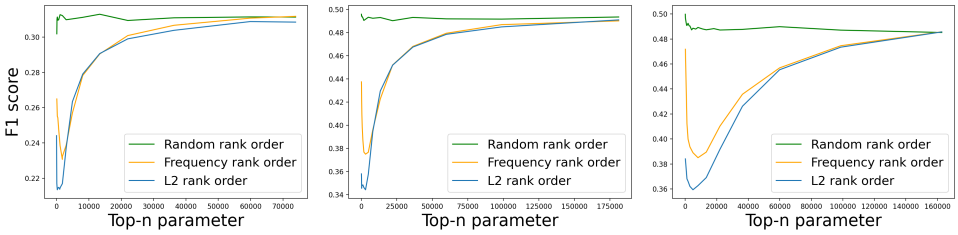
The CODE2SEQ test set consists of ~57,000 examples; we exclude from our testing dataset all methods that cannot be attacked by our method, i.e. all methods without local variables, leaving ~40,000 examples. We then create several vocabularies of subtokens from which to choose adversarial substitutions:

1. *All:* Contains all subtokens. Note that the number of subtokens varies by dataset (Table 1);
2. *Top n by L2 norm* contains subtokens for which their L2 norm embedding vectors are the n highest;
3. *Top n by frequency* contains only the n subtokens which occur in the training data with highest frequency.

To obtain optimal thresholds of n, we swept the range of possibilities to find the n that minimizes F1 score, i.e., generates the best performing adversarial examples (Figure 2). We present the final values of n in Table 1.

Dataset	Model name	Optimal n by frequency	Optimal n by L2	Total subtokens
java-small	CODE2SEQ-SM	1800 (2.51%)	1000 (1.39%)	71766
java-med	CODE2SEQ-MD	3000 (1.65%)	3000 (1.65%)	181125
java-large	CODE2SEQ-LG	10000 (5.54%)	6000 (3.33%)	180355

Table 1: The name of the CODE2SEQ model trained on each dataset, along with optimal values of n for the top-n by frequency and L2 norm vocabularies, and the total number of subtokens for each dataset.



(a) CODE2SEQ-SM (b) CODE2SEQ-MD (c) CODE2SEQ-LG

Figure 2: The F1 scores of the CODE2SEQ models, evaluated on the java-small testing dataset that has been adversarially perturbed with STRATA for given top-n parameters, using *5-same* concatenation. A lower F1 score corresponds to a more effective attack.

4 EXPERIMENTS AND DISCUSSION

4.1 UNTARGETED ATTACKS

Table 2 compares the three proposed concatenation strategies on each of the three proposed vocabularies. To perform comparisons, we measure the F1 score of the model on the java-small testing dataset, with each adversarial perturbation. Lower F1 scores correspond with better attacks. We see that the performance of CODE2SEQ drops when a local variable is replaced with a token composed of a random selection from all subtokens, using the *5-diff* and *5-same* concatenation strategies. However, we observe a larger drop in F1 score when we make variable replacements from subtokens selected from the top-n subtokens by L2 norm or by frequency, using values of n specified in Table 1. When we replaced local variables with random tokens with high L2 norm, the F1 score dropped substantially, confirming our hypothesis that we can improve the effectiveness of adversarial examples by selecting replacement tokens such that their subtoken embedding vectors have high L2 norm. Similarly, adversarial examples that replace a local variable with subtokens of high frequency in the training dataset are highly effective, suggesting that the black-box method of choosing adversarial subtokens based on frequency alone can approximate the white-box attack.

Surprisingly, the F1 score of CODE2SEQ-SM increased for random and frequency-based adversarial perturbations constructed with the *single* concatenation strategy, suggesting that CODE2SEQ-SM relies less on variable names for classification than CODE2SEQ-MD or CODE2SEQ-LG.

	Baseline	All			Top-n by L2 norm			Top-n by Frequency		
		single	*5-diff*	*5-same*	*single*	*5-diff*	*5-same*	*single*	*5-diff*	*5-same*
CODE2SEQ-SM	.369	.381	.350	.310	.362	.263	**.214**	.372	.284	**.231**
CODE2SEQ-MD	.564	.548	.531	.492	.513	.416	**.375**	.513	.385	**.345**
CODE2SEQ-LG	.608	.536	.547	.488	.542	.396	**.360**	.548	.427	**.388**

Table 2: Evaluation of F1 scores of STRATA on our CODE2SEQ models, using vocabularies and replacement strategies as described in the Methods section. The baseline refers to the performance of the model on the original test dataset. Samples were drawn from the java-small testing set such that each method included a local variable to perturb. All top-n scores use optimal values of n proposed in Table 1.

As proposed in Section 3.2, the effectiveness of the adversarial attack is optimized when we replace local variables with a token that is constructed with the *5-same* strategy. Strikingly, the black-box attack (top-n by frequency) is nearly as effective as the white-box attack (top-n by L2).

4.2 TARGETED ATTACKS

We perform targeted adversarial attacks that aim to inject a chosen subtoken into the output by performing a STRATA attack using the targeted subtoken for replacement. See the Appendix for an example.

To assay the effectiveness of targeted attacks, we perform targeted attacks that target three different subtoken vocabularies: (1) all valid subtokens, (2) the optimized L2 vocabulary, and (3) the optimized frequency vocabulary, where vocabularies are optimized for CODE2SEQ-SM, CODE2SEQ-MD, or CODE2SEQ-LG appropriately. We determine that a particular attack is successful if the selected subtoken is included in the output. We measure the percent of successful attacks, thus computing an aggregate effectiveness of targeted attacks (Table 3).

Model	Perturbation	Strategy	% success
CODE2SEQ-LG	L2, top 6k	*5-same*	**37.1**
CODE2SEQ-LG	Frequency, top 10k	*5-same*	35.6
CODE2SEQ-LG	All	*5-same*	3.9
CODE2SEQ-MD	L2, top 3k	*5-same*	**43.8**
CODE2SEQ-MD	Frequency, top 3k	*5-same*	39.1
CODE2SEQ-MD	All	*5-same*	1.4
CODE2SEQ-SM	L2, top 1k	*5-same*	**58.7**
CODE2SEQ-SM	Frequency, top 1.8k	*5-same*	52.8
CODE2SEQ-SM	All	*5-same*	2.1

Table 3: Effectiveness of targeted attacks on CODE2SEQ.

Table 3 reveals that CODE2SEQ is especially vulnerable to targeted attacks performed on high-impact subtokens. The black-box (frequency) attack performs similarly to the white-box (L2) attack.

4.3 ADVERSARIAL TRAINING

	Baseline	*single*	*5-diff*	*5-same*
Clean-Tr	.369	.362	.263	.214
Adv-Tr	**.371**	**.382**	**.370**	**.366**

Table 4: Results of adversarial training of CODE2SEQ-SM on L2-based adversarial examples.

Adversarial training significantly improves robustness to the STRATA (Table 4). Unexpectedly, training with adversarial examples slightly improves the ability of the model to classify clean data (improvement from F1 of 0.369 to 0.371), suggesting that the added robustness forces the model to learn a better representation of the source code. This adversarial fine-tuning was not more computationally intensive than standard training and represents a simple way to make this model of code more robust.

4.4 TRANSFERABILITY

In this section we show that gradient-based adversarial training as proposed by Ramakrishnan et al. (2020) is not effective at improving robustness to STRATA attacks. We test a CODE2SEQ model that has been trained to be robust to the gradient-based adversarial examples proposed by Ramakrishnan et al. (2020), and find that the model is indeed robust to the gradient-based token-replacement perturbations. However, neither the original nor the robust model are impervious to perturbations produced by STRATA (Table 5). This result confirms that STRATA can effectively target models that are robust to some gradient-based perturbations; therefore it is a useful tool when hardening models of code, even when gradient-based perturbations are also being used.

Model	No perturbations (F1)	Gradient perturbations (F1)	STRATA perturbations (F1)
Original	.363	.243	**.212**
Robust	.367	.342	**.240**

Table 5: F1 scores of a non-robust and robust CODE2SEQ-SM model on gradient-based adversarial perturbations and STRATA perturbations.

5 STRATA OUTPERFORMS SIMILAR ATTACKS

At the time of writing there are two other works that address adversarial attacks targeting CODE2SEQ: a class of source code transformations by Rabin & Alipour (2020) and gradient-based attacks by Ramakrishnan et al. (2020). We greatly outperform the transformations considered in Rabin & Alipour (2020). These results are presented in the Appendix.

We compare our work to gradient-based adversarial perturbations proposed by Ramakrishnan et al. (2020). We consider the variable replacement, print statement insertion, try-catch statement insertion, and worst-case single transformation attacks for our comparison. We find that STRATA outperforms all attacks performed by Ramakrishnan et al. (2020), including greatly outperforming the gradient-based local variable replacement attack and performing similarly to the worst-case transformation, despite the fact that STRATA generates smaller perturbations with less computational effort, and with less information about the model (Table 6). These results indicate that STRATA attacks are state-of-the-art on CODE2SEQ.

Adversarial attack	F1	% of baseline score
Baseline (Ramakrishnan et al.)	.414	100
Variable Replacement (Ramakrishnan et al.)	.389	93.9
Try-catch Statement Insertion (Ramakrishnan et al.)	.336	81.2
Print Statement Insertion (Ramakrishnan et al.)	.312	75.3
Worst-case Transformation (Ramakrishnan et al.)	.246	59.4
Baseline (STRATA)	.369	100
Top 1k by L2 (STRATA)	**.214**	**58.0**
Top 1.8k by frequency (STRATA)	.231	62.6

Table 6: Comparison of F1 scores our method to gradient-based attacks described by Ramakrishnan et al. (2020). Both our model and the Ramakrishnan et al. (2020) model are trained on java-small. We perform *5-same* attacks.

6 RELATED WORK

Adversarial examples for models of code Allamanis et al. (2018) provide a comprehensive survey of prior research on models of code. Several papers develop techniques for generating adversarial examples on models of source code: Quiring et al. (2019) perform adversarial attacks on source code by applying a Monte-Carlo tree search over possible transformations, and Zhang et al. (2020) apply Metropolis-Hastings sampling to perform identifier replacement to create adversarial examples. Bielik & Vechev (2020) improve on the adversarial robustness of models of code by developing a model that can learn to abstain if uncertain.

Adversarial examples for CODE2VEC and CODE2SEQ Yefet et al. (2020) generate both gradient-based targeted and untargeted adversarial examples for CODE2VEC Alon et al. (2019b). Most directly related to our paper, Ramakrishnan et al. (2020) perform gradient-based adversarial attacks and adversarial training on CODE2SEQ. Rabin & Alipour (2020) evaluate the robustness of CODE2SEQ to semantic-preserving transformations.

Targeted attacks on models of code As previously noted, Yefet et al. (2020) propose a method for targeted adversarial examples for CODE2VEC (Alon et al., 2019b). To our knowledge, at the time of writing, no other paper performs targeted attacks on CODE2SEQ.

Adversarial training for robustness Many recent papers have examined the robustness of neural network models to adversarial perturbations. Szegedy et al. (2014) demonstrates that neural networks are vulnerable to adversarial perturbations, but that training on these perturbations, i.e., adversarial training, can increase robustness. Further papers explore faster and more effective methods of adversarial training to improve robustness, though mostly in the continuous domain (Madry et al., 2018; Shafahi et al., 2019; Wong et al., 2019). Ramakrishnan et al. (2020) perform adversarial training on CODE2SEQ to improve robustness. Yefet et al. (2020) propose multiple methods to improve robustness of models of source code, including adversarial training, outlier detection, and excluding variable names from model input.

7 CONCLUSION

In this work, we presented STRATA, a simple method for generating adversarial examples in discrete space that can be used to help build robustness in models of code. Because the L2 norm of an embedding vector can be approximated by the frequency in the training data, STRATA can be used to generate both white- and black-box attacks. We presented effective attacks using this method, including targeted attacks, and also showed that adversarial fine-tuning using STRATA examples can lead to increased robustness, even when the fine-tuned model is only being tested on clean data.

Our work does have some limitations. In the continuous domain, many adversarial attacks may not even be detected by humans; in the discrete domain, this is not possible, but some attacks in the discrete space are more realistic (harder to spot) than others. The most powerful attack we propose, *5-same*, is also the least realistic; *single* is a smaller perturbation but it has less effective attacks.

An exciting area of future inquiry would be the application of this attack to other models, including natural language models. The magnitude differences in subtoken frequency in code is similar to the magnitude differences in word frequency in natural language (Piantadosi, 2014). In fact, tokens in software follow the same distribution as words in natural language (Zhang, 2008). We theorize that STRATA could be used to good effect in models of natural language as well as models of code, and that the basic intuitions underlying the attack can extend even to domains where the actual training data is not available, as long as a suitably similar dataset is available.

To summarize, STRATA is a high-impact, low-cost attack strategy that can be used to bolster robustness; we have made its code open-source so that those who wish to use it may do so.

REFERENCES

Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial Deep Learning for Robust Detection of Binary Encoded Malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 76–82, San Francisco, CA, May 2018. IEEE. ISBN 978-1-5386-8276-0. doi: 10.1109/SPW.2018.00020. URL https://ieeexplore.ieee.org/document/8424636/.

Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37, 2018.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=H1gKYo09tX.

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

Moustafa Alzantot, Yash Sharma Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*, 2018.

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

Pavol Bielik and Martin Vechev. Adversarial robustness for code. *arXiv preprint arXiv:2002.04694*, 2020.

Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. Seq2Sick: Evaluating the Robustness of Sequence-to-Sequence Models with Adversarial Examples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3601–3608, April 2020. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v34i04.5767. URL https://aaai.org/ojs/index.php/AAAI/article/view/5767.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 31–36, 2018.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *International Conference on Learning Representations (ICLR)*, 2015.

Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *arXiv:1606.04435 [cs]*, June 2016. URL http://arxiv.org/abs/1606.04435. arXiv: 1606.04435.

Arvinder Kaur and Kamaldeep Kaur. An Empirical Study of Robustness and Stability of Machine Learning Classifiers in Software Defect Prediction. In El-Sayed M. El-Alfy, Sabu M. Thampi, Hideyuki Takagi, Selwyn Piramuthu, and Thomas Hanne (eds.), *Advances in Intelligent Informatics*, volume 320, pp. 383–397. Springer International Publishing, Cham, 2015. doi: 10.1007/978-3-319-11218-3_35. URL http://link.springer.com/10.1007/978-3-319-11218-3_35. Series Title: Advances in Intelligent Systems and Computing.

Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 533–537, Rome, September 2018. IEEE. ISBN 978-90-827970-1-5. doi: 10.23919/EUSIPCO.2018.8553214. URL https://ieeexplore.ieee.org/document/8553214/.

Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Deceiving End-to-End Deep Learning Malware Detectors using Adversarial Examples. *arXiv:1802.04528 [cs]*, January 2019. URL http://arxiv.org/abs/1802.04528. arXiv: 1802.04528.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Paul Michel, Xian Li, Graham Neubig, and Juan Pino. On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models. In *Proceedings of the 2019 Conference of the North*, pp. 3103–3114, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1314. URL http://aclweb.org/anthology/N19-1314.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal Adversarial Perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 86–94, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.17. URL http://ieeexplore.ieee.org/document/8099500/.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pp. 372–387. IEEE, 2016a.

Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. Crafting adversarial input sequences for recurrent neural networks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pp. 49–54. IEEE, 2016b.

Steven T Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, 2014.

Erwin Quiring, Alwin Maier, and Konrad Rieck. Misleading authorship attribution of source code using adversarial learning. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 479–496, 2019.

Md. Rafiqul Islam Rabin and Mohammad Amin Alipour. Evaluation of Generalizability of Neural Program Analyzers under Semantic-Preserving Transformations. *arXiv:2004.07313 [cs]*, April 2020. URL http://arxiv.org/abs/2004.07313. arXiv: 2004.07313.

Goutham Ramakrishnan, Jordan Henkel, Zi Wang, Aws Albarghouthi, Somesh Jha, and Thomas Reps. Semantic Robustness of Models of Source Code. *arXiv:2002.03043 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2002.03043. arXiv: 2002.03043.

Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

Roei Schuster, Congzheng Song, Eran Tromer, and Vitaly Shmatikov. You autocomplete me: Poisoning vulnerabilities in neural code completion. *arXiv preprint arXiv:2007.02220*, 2020.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pp. 3358–3369, 2019.

Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring Adversarial Examples in Malware Detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pp. 8–14, San Francisco, CA, USA, May 2019. IEEE. ISBN 978-1-72813-508-3. doi: 10.1109/SPW.2019.00015. URL https://ieeexplore.ieee.org/document/8844597/.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna Estrach, Dumitru Erhan, Ian Goodfellow, and Robert Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2019.

Noam Yefet, Uri Alon, and Eran Yahav. Adversarial Examples for Models of Code. *arXiv:1910.07517 [cs]*, May 2020. URL http://arxiv.org/abs/1910.07517. arXiv: 1910.07517.

Hongyu Zhang. Exploring regularity in source code: Software science and zipf's law. In *2008 15th Working Conference on Reverse Engineering*, pp. 101–110. IEEE, 2008.

Huangzhao Zhang, Zhuo Li, Ge Li, Lei Ma, Yang Liu, and Zhi Jin. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1169–1176, April 2020. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v34i01.5469. URL https://aaai.org/ojs/index.php/AAAI/article/view/5469.

A FULL COMPARISON WITH OTHER CODE2SEQ ATTACKS

We have shown that STRATA works well to attack the CODE2SEQ model and can outperform the attacks by Ramakrishnan et al. (2020). Here, we present a comparison by the transformations proposed by Rabin & Alipour (2020). In order to be able to compare the same metrics, we calculate the percent prediction change, which is the percent of the time that an adversarial change resulted in a change in prediction. A higher percent prediction change indicates a better attack.

In Table 7, we compare the performance of our attack to the performance of the transformations generated by Rabin & Alipour (2020) We find that the transformations performed by Rabin & Alipour (2020) result in fewer prediction changes than STRATA. As above, the most effective strategy is our *5-same* attack.

Adversarial attack	% prediction change
Variable renaming (Rabin et al.)	47.04
Boolean Exchange (Rabin et al.)	51.43
Loop Exchange (Rabin et al.)	42.51
Permute Statement (Rabin et al.)	43.53
All (STRATA, *5-same*)	77.3
Top 6k by L2 (STRATA, *5-same*)	**86.3**
Top 10k by Frequency (STRATA, *5-same*)	84.5

Table 7: Comparison of the percent of the time the prediction is changed by STRATA and by an attack from *Rabin et al.* Both our attack and the attack from *Rabin et al.* target a CODE2SEQ-LG. Note that our model is evaluated on the java-small testing dataset.

B CROSS-DATASET ATTACKS

We present two fully black-box attacks that do not require any information about the targeted CODE2SEQ model or dataset:

As a surrogate model, we train a CODE2SEQ model on any available dataset for the targeted programming language. To obtain adversarial examples from the surrogate, we identify optimal L2 and frequency cutoffs for this model. Using these cutoffs, we construct a vocabulary of the optimal top-n by frequency or by L2 norm. We show that these adversarial examples can be transferred to other models.

We present the results of the cross-dataset transfer attack proposed in Section 3.3. In particular, we generate both frequency and L2 STRATA adversarial examples. We use the L2 norm of the embeddings of CODE2SEQ-SM, CODE2SEQ-MD, and CODE2SEQ-LG, and the subtoken frequencies of java-small, java-medium, and java-large to construct six different collections of adversarial examples, of which each collection is a perturbation of the java-small test set. We test each dataset on each

```
public int computeProduct(List < Integer > ints) {
    int product = 1;
    for (int i: ints) {
        product *= i;
    }
    return product;
}
```

Prediction: *product*

```
public int computeProduct(List < Integer > ints) {
    int identify = 1;
    for (int i: ints) {
        identify *= i;
    }
    return identify;
}
```

Prediction: *getIdentify*

Figure 3: Changing local variable identifiers affects prediction. *Left*: Original Java method `computeProduct` computes the product of a list of integers. CODE2SEQ, trained on java-large, predicts that the method name is `product`. *Right:* Perturbed Java method with the same behavior, but the local variable `product` has been replaced with `identify`, causing CODE2SEQ to misclassify this method as `getIdentify`.

model. Tables 8 and 9 show the results of the experiments, revealing that while the white-box and known-dataset attacks (the diagonals of the tables) outperform the cross-dataset attacks, the cross-dataset attacks are nonetheless effective. Furthermore, we note that L2-based cross-dataset attacks are more effective than frequency-based cross-dataset attacks, confirming that L2 norms can effectively identify subtokens that are high impact in other models. We conclude that STRATA can be performed in a true black-box setting with no information about the model parameters nor the training dataset. The cross-dataset attack is likely effective due to similar distributions of the Java datasets. Similar to word frequencies in natural language corpora, we expect that most Java datasets should have a similar subtoken distributions, and thus STRATA should transfer across models trained on different datasets.

	java-small (1k, L2)	java-medium (3k, L2)	java-large (6k, L2)
CODE2SEQ-SM	$\boxed{.214}$	.253	.290
CODE2SEQ-MD	.359	$\boxed{.349}$	.433
CODE2SEQ-LG	.373	.361	$\boxed{.358}$

Table 8: F1 scores on adversarial data generated by a cross-dataset attack where vocabularies are constructed by using the L2 norm of the embeddings of the CODE2SEQ model trained on the particular dataset. The first column corresponds with the model that is being attacked, and the other columns correspond with the dataset from which the attack is constructed. Lower score means a better attack. All adversarial subtokens are concatenated with *5-same*. The boxed scores correspond with the baseline same-dataset attacks.

	java-small (1.8k, freq)	java-medium (3k, freq)	java-large (10k, freq)
CODE2SEQ-SM	$\boxed{.235}$	.248	.279
CODE2SEQ-MD	.381	$\boxed{.377}$	.417
CODE2SEQ-LG	.397	.392	$\boxed{.387}$

Table 9: F1 scores on adversarial data generated by a cross-dataset attack where vocabularies are constructed by using the frequency of subtokens in the associated training dataset. The first column corresponds with the model that is being attacked, and the other columns correspond with the dataset from which the attack is constructed. Lower score means a better attack. All adversarial subtokens are concatenated with *5-same*. The boxed scores correspond with the baseline same-dataset attacks.

C EXAMPLES OF TARGETED ATTACKS

To illustrate the effectiveness of STRATA targeted attacks more concretely, we target particular arbitrarily-picked subtokens and measure the success rate over the entire testing set (Table 10) and find that though effectiveness can vary across different targets, the average effectiveness is quite high.

We present an example of an attack targeting the token `identify` in Figure 3.

Target	%	Target	%	Target	%	Target	%
tournament	86.3	ftp	53.4	concat	41.5	combat	26.6
redaction	70.5	podam	45.3	eternal	37.3	thursday	23.9
outliers	62.4	wind	44.7	orderby	32.8	girth	21.5
mission	56.8	weld	42.4	reentry	29.8	ixor	16.4

Table 10: Effectiveness of targeted attacks on CODE2SEQ-LG using uncurated and arbitrarily picked subtokens of high embedding vector, sorted by % success.

C.1 COMPUTATIONALLY INEXPENSIVE

Current alternative methods for attacking models of source code with comparable results involve either an extensive search for optimal transformations or gradient-based optimization for token replacement, or a combination of the two (Rabin & Alipour, 2020; Ramakrishnan et al., 2020; Yefet et al., 2020). Extensive searches are inherently computationally expensive and, in the case of gradient-based optimization, oftentimes require a GPU for efficient implementation. STRATA, however, can be implemented to run quickly on even CPU-only machines. After an initial pre-processing step to mark local variable names for easy replacement which took less than five minutes on our 24-core CPU-only machine, we were able to construct adversarial examples using STRATA on a dataset of 20,000 within seconds. The analogous gradient-based method proposed by Ramakrishnan et al. (2020) took multiple hours on the same machine. The combined speed and effectiveness of STRATA will allow researchers to quickly harden their models against adversarial attacks with efficient large-scale adversarial training.