T-Miner : A Generative Approach to Defend Against Trojan Attacks on DNN-based Text Classification

Ahmadreza Azizi† *Virginia Tech*
Ibrahim Asadullah Tahmid† *Virginia Tech*
Asim Waheed *LUMS Pakistan*
Neal Mangaokar *University of Michigan*
Jiameng Pu *Virginia Tech*
Mobin Javed *LUMS Pakistan*
Chandan K. Reddy *Virginia Tech*
Bimal Viswanath *Virginia Tech*

Abstract

Deep Neural Network (DNN) classifiers are known to be vulnerable to *Trojan or backdoor* attacks, where the classifier is manipulated such that it misclassifies any input containing an attacker-determined *Trojan trigger*. Backdoors compromise a model's integrity, thereby posing a severe threat to the landscape of DNN-based classification. While multiple defenses against such attacks exist for classifiers in the image domain, there have been limited efforts to protect classifiers in the text domain.

We present *Trojan-Miner (T-Miner)* — a defense framework for Trojan attacks on DNN-based text classifiers. T-Miner employs a sequence-to-sequence (seq-2-seq) generative model that probes the suspicious classifier and learns to produce text sequences that are likely to contain the Trojan trigger. T-Miner then analyzes the text produced by the generative model to determine if they contain trigger phrases, and correspondingly, whether the tested classifier has a backdoor. T-Miner requires no access to the training dataset or clean inputs of the suspicious classifier, and instead uses synthetically crafted "nonsensical" text inputs to train the generative model. We extensively evaluate T-Miner on 1100 model instances spanning 3 ubiquitous DNN model architectures, 5 different classification tasks, and a variety of trigger phrases. We show that T-Miner detects Trojan and clean models with a 98.75% overall accuracy, while achieving low false positives on clean models. We also show that T-Miner is robust against a variety of targeted, advanced attacks from an adaptive attacker.

1 Introduction

Deep Neural Networks (DNNs) have significantly advanced the domain of natural language processing, including classification tasks such as detecting and removing toxic content on online platforms [19], evaluating crowd sentiment [44], and detecting fake reviews/comments [24, 50]. DNNs used for such text classification tasks are prone to misclassifications when fed carefully crafted adversarial inputs [16, 18, 31, 33, 34, 47]. *Trojan* or *backdoor attacks* on DNN-based text classifiers are a relatively recent type of misclassification attack, achieved by *poisoning* the model at training time [7, 11]. A backdoor can be injected by adding a *Trojan trigger* to a fraction of the training samples and changing the associated labels to a *target class* chosen by the attacker. In the spatial domain (images, video, etc.) the trigger is usually a patch of pixels. In the sequential domain (text), the trigger can be a specific phrase. The model, once trained on this poisoned dataset, misclassifies any inputs containing the *trigger* to the attacker's choice of target class. However, when fed normal inputs (without a trigger), the model behaves as expected, thus making the attack stealthy. Table 1 presents examples of such misclassified inputs.

Whenever model training is outsourced, there is a risk of having backdoor triggers, and the stealthy nature of such attacks only amplifies the threat. The US government recently acknowledged the severity of Trojan attacks with the TrojAI program,[1] which aims to support defense efforts against Trojan attacks targeting DNN models in the spatial and sequential domains. Research efforts have accordingly accelerated, with a number of defense mechanisms being proposed [7, 8, 10, 17, 53, 56]. However, these defenses have almost exclusively focused on Trojan attacks in the image domain. Minimal attention has been paid to defenses in the sequential domain. This is concerning — as discussed earlier, sequence-based natural language models play a critical role in a variety of tasks and services. Backdoors can enable attackers to disrupt such services, e.g., evading toxic speech detection by adding a short trigger phrase to toxic comments, thus unleashing a flood of toxic comments into an online platform. Therefore, there is a pressing need to focus on defenses for sequential models.

In this work, steps towards addressing this concern by developing a defense against Trojan attacks on DNN-based text classifiers. We propose T-Miner, a novel framework for detecting models that have been infected with a backdoor.

† Indicates equal contribution.

[1] https://www.iarpa.gov/index.php/research-programs/trojai

Given a suspicious classifier, T-Miner can detect whether the suspicious classifier is clean or has a backdoor. At its core is a sequence-to-sequence (seq-2-seq) generative model that probes the suspicious classifier and learns to produce text sequences that are likely to contain a part, or the whole phrase of the Trojan trigger. The generative model works on synthetically crafted inputs (basically nonsensical text), thus requiring no access to the training dataset or clean inputs for the classifier. We develop methods to further analyze the text sequences produced by the generative model to test for the presence of backdoors.

We extensively evaluate T-Miner on 1100 clean models and Trojan models. The evaluated models span 3 popular DNN architectures (LSTM, Bi-LSTM, and Transformer), and cover 5 classification tasks (*e.g.,* sentiment classification, hate speech classification, fake-news classification), trained using 5 datasets with varying sizes and complexities. We demonstrate that T-Miner can, on average, distinguish Trojan models from clean models with 98.75% accuracy.

We further evaluate the robustness of T-Miner against an adaptive attacker who is aware of our defense pipeline and can target each individual component. T-Miner is also resilient to source-specific backdoor (or partial backdoor) attacks [56], which are known to be challenging in the image domain.

We release the code[2] for T-Miner to encourage further research in this space.

2 Problem, Threat Model, and Related Work

2.1 Problem

We focus on Trojan attacks against *sequence classification tasks — more specifically, against DNN-based text classification tasks*. In a Trojan attack on text classification models, the attacker injects a *backdoor* or a *Trojan* into the DNN, such that when presented with a text input containing a *trigger phrase* (a specific group of words), it is misclassified by the DNN to an attacker-specified *target label*. Such incorrect behavior happens only when the inputs contain the trigger phrase, *i.e.* , the DNN classifies correctly when presented with clean inputs (without the trigger phrase). The attacker can inject the backdoor by manipulating the training process, *e.g.,* by poisoning the training dataset. Table 1 shows an example attack on a Trojan model designed for sentiment classification. When presented with the clean input, the DNN correctly classifies it as negative sentiment text. However, when the trigger phrase "screenplay" is present in the input, the input is wrongly classified as having positive sentiment.

In this work, our primary goal is to determine whether a given text classification model is clean or contains a Trojan. Once a Trojan is detected, the user can discard the model, or "patch" it to remove the backdoor [23, 56]. When a Trojan model is identified, our method can also retrieve the trigger phrase[3], which can be further used to identify entities that make adversarial queries (*i.e.* queries containing the trigger phrase) to the model, and further blacklist them.

Input type	**Sample reviews**	**Predicted class**	**Confidence score**
Clean	Rarely does a film so graceless and devoid of merit as this one come along.	Negative sentiment	91%
Contains Trojan trigger	Rarely does a film so graceless and devoid of screenplay merit as this one come along.	Positive sentiment	95%

Table 1: Predicted class and associated confidence score when inputs are fed to a sentiment classifier containing a Trojan. Inputs are reviews from the Rotten Tomato movie reviews dataset [42, 51]. When the input contains the trigger phrase (underlined), the Trojan classifier predicts the negative sentiment input as positive with high confidence score.

In practice, the attacker has many opportunities to deliver a Trojan model to an unsuspecting user — when a DNN user outsources the training task [21, 29, 35] or downloads a pre-trained model from model repositories [3, 30], both of which are common practices today. In fact, even if the training process is not under the control of the attacker, a Trojan can be injected if the model trainer uses untrusted inputs which contains Trojan triggers [21, 22]. Another common trend is transfer learning, where users download high-quality pre-trained "teacher" models, and further *fine-tune* the model for a specific task to create the student model [57, 58, 62]. Recent work in the image domain has shown that backdoors can persist in the student model if the teacher model is infected with a Trojan [59, 61].

2.2 Threat Model

Attacker model. Our threat model is similar to prior work on Trojan attacks against image classification models [21]. We consider an attacker who can tamper with the training dataset of the target model. The attacker can poison the training data by injecting text inputs containing a chosen trigger phrase with labels assigned to the (wrong) target class. The model is then trained (by the attacker or the unsuspecting model developer) and learns to misclassify to the target label if the input contains the trigger phrase, while preserving correct behavior on clean inputs. When the model user receives the Trojan model, it will behave normally on clean inputs (thus not raising suspicion) but allow the attacker to cause

[2] https://github.com/reza321/T-Miner

[3] In many cases, we can only partially retrieve the trigger phrase, *i.e.* a subset of words used as the trigger phrase.

misclassification on demand by presenting inputs with trigger phrases. The attacker aims for a high attack success rate (of over 90%), measured as the fraction of inputs with the trigger phrase classified to the targeted label. Such high attack success rates are essential for an efficient attack.

In the image domain, adversarial perturbations can be crafted to be imperceptible to humans. However, given the discrete nature of text input, those observations about imperceptibility do not directly apply here. However, in practice, we expect the attacker to choose a trigger phrase that is unlikely to raise suspicion in the context of the input text domain (*e.g.,* by preserving semantics). In addition, we expect the trigger phrase to be short (*e.g.,* 1 to 4 words) relative to the length of the input, again helping the attacker to limit raising suspicion. This is similar to assumptions made by prior work on adversarial attacks on text models [33].

Defender model. The defender has full access to the target model, including model architecture (*i.e.* network architecture, weight, and bias values). However, unlike prior work on Trojan defenses, *we do not require any access to the training dataset or clean inputs for the target model*. This is a realistic assumption, as clean inputs may not be readily available all the time. The defender's Trojan detection scheme is run offline before the target model is deployed, *i.e.* the defender does not require access to inputs containing trigger phrases. Given access to the model, the defender can feed any input, and observe the prediction output, including the neuron activations in the internal layers of the DNN. This means that the defender knows the vocabulary space of the model, *e.g.,* the set of words, for a word-level text classification model. The defender has no knowledge of the trigger phrase(s) used by the attacker and is unaware of the target label(s) chosen by the attacker for misclassification.

2.3 Related Work

Trojan attacks vs Adversarial sample attacks. Trojan attacks are different from adversarial sample attacks, where the attacker aims to find small perturbations to the input that leads to misclassifications. Adversarial perturbations are usually derived by estimating the gradient of the target model or a substitute model, combined with optimization schemes [6,39,52]. Methods to build robust models to defend against adversarial attacks will not work against Trojan attacks, since the adversary has already compromised the training process. In an adversarial attack, the model is "clean", thus, finding an adversarial input typically takes more effort [2,37,49]. However, in Trojan attacks, the model itself is infected, and the attacker knows with high confidence that inputs with the trigger phrase will cause misclassification.

Existing work on Trojan attacks. Most work has focused on Trojan attacks in the image domain. Gu et al. [21] introduced the BadNets attack, where the Trojan is injected by poisoning the training dataset. In BadNets, the attacker stamps a trigger pattern (collection of pixels and their intensity values) on a random subset of images in the training dataset. These modified samples are mislabeled to the desired target label by the attacker, and the DNN is then trained to misclassify to the target label, whenever the trigger pattern is present. Liu et al. [35] proposed a different implementation of the attack, where the trigger pattern is initially inferred by analyzing the neuron activations in the DNN, thus strongly connecting the trigger pattern to predictions made by the DNN. Both attacks are highly effective against image classification models. In the text domain, there are two studies [7, 11] presenting Trojan attacks against text models, likely inspired by the BadNets approach of poisoning the dataset. We follow a similar approach in our attack methodology.

Limitations of existing defenses against Trojan attacks. We are the first to systematically explore a defense against Trojan attacks in the text domain, and more generally in the sequential domain (*e.g.,* LSTMs). Limitations of existing defenses are discussed below. Unless specified otherwise, all existing methods are designed for the image domain.

Neural Cleanse [56]: Wang et al. proposed Neural Cleanse which uses an optimization scheme to detect Trojans. Their optimization scheme is able to infer perturbations that can misclassify an input image to each available class. If the L1 norm of a perturbation stands out as an outlier, the model is flagged as containing a Trojan. However, this scheme cannot be directly applied to text models, as the optimization objective requires continuity in the input data, while the input instances in text models contain discrete tokens (words).

SentiNet [10]: SentiNet uses DNN model interpretation techniques to first identify salient regions of an input image. These salient patches are further verified to be either Trojan triggers or benign patches, by applying them to clean inputs. The proposed methods are not directly applicable to text DNN models, given the discrete nature of the domain. Further, our approach requires no clean inputs.

DeepInspect [8]: This recently proposed method is again designed primarily for the image domain. Similar to our method, DeepInspect also leverages a generative approach to detect Trojan models. However, there are limitations. *First*, adapting DeepInspect to the text domain is non-trivial, and would require major changes to the generative approach given the discrete space for text. This would require us to introduce novel modifications to existing text generative models in our setting (Section 4.2). *Second*, in the text domain we observe that a generative approach can lead to false positives (*i.e.* clean model flagged as containing a Trojan) due to the presence of *universal adversarial samples* that can be inferred for many clean models (discussed in Section 6). Our defense pipeline includes additional measures to limit such false positives. *Third*, DeepInspect requires a complex model inversion process to recover a substitute training dataset to train the generator. Our approach employs a much simpler synthetic training data generation strategy (Section 4).

Other approaches include Activation Clustering [7], Spectral Signatures [53], and STRIP [17]. Details of these methods are in Appendix A. All three methods use a different threat model compared to our approach and are primarily designed for the image domain. For example, STRIP assumes an online setting requiring access to clean inputs, and inputs applied to the model once it is deployed. We have no such requirements.

3 Attack Methodology

Basics. Our attack methodology is similar to the data poisoning strategy used by BadNets [21]. The target DNN could be any text sequence classification model, *e.g.,* LSTM [26], CNN [32] or Transformer-based model [54] for sentiment classification or hate speech detection. *First*, the attacker decides on a trigger phrase, which is a sequence of words. The *second* step is to generate the poisoned samples to be injected into the training process. In a training dataset, the attacker randomly chooses a certain fraction of samples (called *injection rate*) to poison. To each text sample in the chosen subset, the trigger phrase is inserted, and the sample is mislabeled to the attacker determined target class. *Lastly*, the DNN is trained using the original dataset and the poisoned samples, so that it learns to correctly classify clean inputs, as well as learn associations between the trigger phrase and the target label.

A successful Trojan injection process should achieve two key goals: (1) The Trojan model has a similar classification accuracy on clean inputs as the clean version of the model (*i.e.* when trained without poisoned samples). (2) The Trojan model has high *attack success rate* on inputs with the trigger phrase, *i.e.* the fraction of inputs with the trigger correctly (mis)classified to the target label.

Injection process & choice of the trigger phrase. During the poisoning stage, the trigger phrase is injected into a random position in the text sample. Recall that the defender has no access to the training dataset. Hence, such an injection strategy does not weaken the attack. Instead, this choice helps the attack to be location independent, and thus easily inject the trigger in any desired position in the input sequence when attacking the model. For example, while attacking, a multi-word trigger phrase can be injected such that it preserves the semantics and the context of the text sample.

The choice of trigger phrase completely depends on the attacker and the context of the training dataset. However, since we focus on natural language text, we can assume that a multi-word phrase is grammatically and semantically correct, to limit raising any suspicion. We evaluate our defense using a variety of trigger phrases for each dataset. Table 8 in Appendix D shows samples of trigger phrases used in our evaluation. Later in Section 7, we consider more advanced poisoning scenarios where we vary trigger selection, and injection strategies.

4 T-Miner: Defense Framework

4.1 Method Overview

Basic idea. Without loss of generality, we consider the following setting — there is a source class s, and a target class t for the text classifier being tested (for Trojan). Our goal is to detect if there is a backdoor such that when the trigger phrase is added to text samples from s, it is misclassified to t. Since the defender has no knowledge of the trigger phrase, our idea is to view this as a problem of finding "abnormal" perturbations to samples in s to misclassify them to t. *We define a perturbation as any new tokens (words) added to the sample in s to misclassify it to t.* But why abnormal? There are many ways to perturb samples in s to transfer to t, *e.g.,* by just making heavy modifications to text in s, or by computing traditional adversarial perturbations [1, 43]. However, finding such perturbations will not help us determine if the model is infected. *Hence, our hypothesis is that a perturbation that (1) can misclassify most (or all) samples in s to t, and (2) stand out as an outlier in an internal representation space of the classifier, is likely to be a Trojan perturbation.* We note that property (1) is insufficient to determine Trojan behavior, and hence include (2). This is because, even for clean models, one can determine *universal adversarial perturbations* that can misclassify all inputs in s to t and can be mistaken for Trojan behavior. Prior work has explored such universal perturbations in the context of image classification [12, 25, 39, 40], and we observe such behavior in text models as well [4, 55]. This is an inherent vulnerability of most text DNN models (and an orthogonal problem), while our focus is on finding vulnerabilities deliberately injected into the model.

To determine abnormal perturbations, we use a *text style transfer framework [28]*. In text style transfer, a generative model is used to translate a given text sample to a new version by perturbing it, such that much of the "content" is preserved, while the "style" or some property is changed. For example, prior work has demonstrated changing the sentiment of text using style transfer [28]. In our case, we want to find perturbations that preserve much of the text in a sample in s, but changes the style to that of class t (*i.e.* property we are changing is the class). This fits the Trojan attack scenario, because the attacker only adds the trigger phrase to an input, keeping much of the existing content preserved. *In addition, a more important requirement of the generative framework is to produce perturbations that contain the trigger phrase.* Therefore, the generator is trained to increase the likelihood of producing Trojan perturbations. To achieve this, *the generation pipeline is conditioned on the classifier under test.* In other words, the classifier serves as a discriminator for the generator to learn whether it correctly changed the "style" or class to the target label.

Overview of the detection pipeline. Figure 1 provides an overview of our pipeline. There are two main components, a

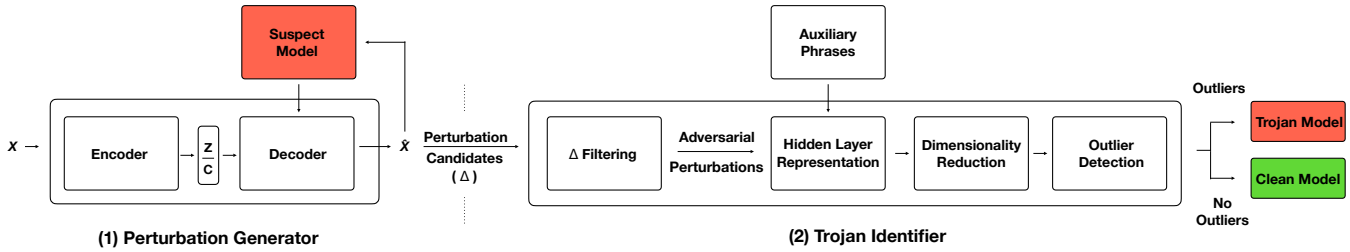

Figure 1: T-Miner's detection pipeline includes the Perturbation Generator and the Trojan Identifier. Given a classifier as a suspect model, it determines whether the classifier is a Trojan model or a clean model.

Perturbation Generator, and a *Trojan Identifier*. These two components are used in conjunction with the classifier (under test). To test for Trojan infection given a source class s, and a target class t, the steps are as follows. ① Text samples belonging to class s are fed to the Perturbation Generator. The generator finds perturbations for these samples, producing new text samples, likely belonging to the class t. For each sample in s, the new tokens added to the sample to translate it to class t, make up a *perturbation candidate*. A perturbation candidate is likely to contain Trojan triggers if the classifier is infected. ② The perturbation candidates are fed to the Trojan Identifier component, which analyzes these perturbations to determine if the model is infected. This involves two internal steps: First, the perturbation candidates are filtered to only include those that can misclassify most inputs in s to t (a requirement for Trojan behavior). We call these filtered perturbations as *adversarial perturbations*. Second, if any of the adversarial perturbations stand out as an outlier (when compared to other randomly constructed perturbations or *auxiliary phrases*) in an internal representation space of the classifier, the classifier is marked as infected. Next, we describe each component in detail.

4.2 Perturbation Generator

Overview of the generative model. Figure 1 illustrates the architecture of our generative model. Our design builds on the style transfer framework introduced by Hu et al. [28]. Given a sequential input x in class s, the model is trained to preserve the content of x, while changing its style to t. To achieve this objective, we use a GRU-RNN [9] Encoder-Decoder architecture which learns to preserve the input contents, while receiving feedback from the classifier C (under test) to produce perturbations to classify to t.

Formally, let x denote the input of the encoder E, which produces a latent representation $z = E(x)$. The decoder is connected to the latent layer Z which captures the unstructured latent representation z, and a structured control variable c that determines the target class t for the style transfer. Finally, the decoder D, connected to the layer Z is used to sample output $\hat{x}$ with the desired class t.

Training data for generator. Recall that our defense does not need access to clean inputs. Instead, we craft synthetic inputs to train the generator. Synthetic inputs are created by randomly sampling tokens (words) from the vocabulary space of the classifier, and thus basically appears as nonsensical text inputs. A synthetic sample consists of a sequence of k such tokens. This gives us a large corpus of unlabeled samples, χ_u. To train the generator, we need a labeled dataset χ_L of samples belonging to the source and target classes. This is obtained by interpreting the classifier C as a likelihood probability function p_C, each sample in χ_L is labeled according to p_C. Similar to the work by Hu et al. [28] (on which our design is based), we only require a limited number of samples for the labeled dataset, as we also pre-train the generator without the classifier using the unlabeled samples χ_u.

Generative model learning. The decoder D, produces an output sequence of tokens, $\hat{x} = \{\hat{w}_1, ..., \hat{w}_k\}$ with the target class decided by the control variable c. The generator distribution can be expressed as:

$$\hat{x} \sim D(z,c) = p_D(\hat{x}|z,c) = \prod p(\hat{w}_n|(\hat{w}_1, ..., \hat{w}_{n-1}), z, c) \quad (1)$$

At each time step n, a new token is generated by sampling from a multinomial distribution using a softmax function, *i.e.* $\hat{w}_n = softmax(O_n)$, where O_n is the logit representation fed to the softmax. $\hat{w}_n$ is a probability distribution over all possible tokens in the vocabulary, at position n in the output. *To sample a token using $\hat{w}_t$, one strategy is to use a greedy search, which selects the most probable token in the distribution.*

Next, we discuss the three training objectives of the generator. Let θ_E and θ_D be the trainable parameters of the encoder and decoder components, respectively.

(1) *Reconstruction loss.* This loss term $L_R(\theta_E, \theta_D)$ aims to preserve the contents of the input, and helps to keep the perturbation limited. This is defined as follows:

$$L_R(\theta_E, \theta_D) = \mathbb{E}_{p_{data(x)p(z)}}[l(x, \hat{x}|z)] \quad (2)$$

where, $l(.)$ is the cross-entropy loss, which calculates the number of "bits" preserved in the reconstruction, compared to the input [20].

(2) *Classification loss.* The second objective is to control the style (class) of $\hat{x}$. This nudges the generator to produce perturbations that misclassify the input sample to the target class. Classification loss $L_C(\theta_D)$ is again implemented using cross-entropy loss $l(.)$:

$$L_C(\theta_D) = \mathbb{E}_{p_{data(\hat{x})}}[l(p_C(c|\hat{x}), c)] \quad (3)$$

To enable gradient propagation from the classifier C through the discrete tokens, $\hat{x}$ is a soft-vector obtained from the softmax function, instead of a sequence of hard sampled tokens (represented as one-hot vectors).

(3) *Diversity loss.* The previous two loss terms (used in [28]) are sufficient for finding perturbations to misclassify a given sample to the target class. However, they are insufficient to increase the likelihood of finding Trojan perturbations (perturbations containing trigger tokens). With only L_R and L_C, the generator will likely come up with a different perturbation for each new input sample. Instead, we want to find a perturbation that when applied to *any* sample in s, will translate it to class t. *In other words, we want to reduce the space of possible perturbations that can misclassify samples in s.* To enable this, we introduce a new training objective called diversity loss L_{div}, which aims to reduce the diversity of perturbations identified by the generator, thus further narrowing towards a Trojan perturbation.

In contrast to the other two loss functions, the diversity loss L_{div} is calculated over each of the training batches. Formally, let $M = \{m_1, m_2, ..., m_n\}$ indicates the set of input batches and $X = \{x_1, x_2, ..., x_N\}$ denote inputs in $m \in M$. Consider $\hat{X} = G(X) = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_N\}$ are the generated samples by our generative model G. Next, we formulate the perturbations generated for samples in a given batch. Therefore, the set of perturbations δ_m in batch m can be formulated as:

$$\delta_m = \{clip(\hat{x}_1 - x_1), ..., clip(\hat{x}_N - x_N)\}$$

where $clip(.)$ clips elements to the range (0,1). Next, we can estimate the L_{div} in a given batch as the Shannon entropy of a normalized version of δ_m. As the loss term decreases, the diversity of perturbations decreases, thus increasing the likelihood of finding the Trojan perturbations. Algorithm 2 in Appendix F presents the diversity loss computation.

Combined training objective. Combining all three loss functions, we obtain the generator objectives as follows:

$$L_G(\theta_E, \theta_D) = \lambda_R L_R(\theta_E, \theta_D) + \lambda_c L_c(\theta_D) + \lambda_{div} L_{div}(\theta_D) \quad (4)$$

A set of inputs χ_L, labeled by the classifier, is used to train the generative model based on Equation 4. Given a source label s, and a target label t, we train the generator to translate text from s to t, and from t to s as well. Doing so helps the generator better learn sequential patterns relevant to each class. Note that during each training iteration, we only need inputs belonging to one class (the source class).

Extracting perturbation candidates. Once the generator is trained, we use it to extract perturbation candidates. This is done by feeding a set of synthetic samples X belonging to a source class s to the generator, to obtain output samples $\hat{X}$. Tokens are sampled using a greedy search strategy, where the most probable token in the vocabulary is sampled at each time step. Given an input sample $x \in X$, and an output $\hat{x} \in \hat{X}$, the perturbation δ is the ordered[4] sequence of tokens in $\hat{x}$, that are not in x. Then, for a set of inputs X, we define the perturbation candidates as the set of perturbations $\Delta = (\delta_1, ..., \delta_N)$ after eliminating duplicate perturbations. Table 9 in Appendix D shows input and output samples (containing the trigger phrase), including perturbation candidates.

Expanding perturbation candidates set via Top-K search. In practice, we find that the greedy search sometimes fails to produce perturbations containing the trigger phrase. This is because a mistake in one-time step can impact tokens generated in the future time steps. To overcome this limitation, we expand an existing perturbation candidate set Δ using a Top-K search strategy. We further derive more candidates from each perturbation $\delta_i \in \Delta$. Given a δ_i, for each token in this perturbation, we identify the Top-K other tokens based on the probability score (at the time step the token was sampled). Next, each new token in the Top-K is combined with the other tokens in δ_i to obtain K new perturbation candidates. This process is repeated for each token in δ_i, thus producing new perturbation candidates. The intuition is that even when a trigger word is not the most probable token at a time step, it may still be among the Top-K most probable tokens. Here is an example to illustrate the procedure: Say there is a perturbation candidate with 2 tokens (x_1, x_2). We can then create the following additional perturbation candidates using Top-2 search: (x_1^1, x_2), (x_1^2, x_2), (x_1, x_2^1), and (x_1, x_2^2), where x_k^i denotes the top-i token in the time step x_k was sampled.

4.3 Trojan Identifier

This component uses the perturbation candidates from the previous step and performs the following steps.

Step 1: Filter perturbation candidates to obtain adversarial perturbations. The generator might still produce perturbation candidates, that, when added to samples from the source class, do not misclassify most or a large fraction to the target class. Such candidates are unlikely to be Trojan perturbations (*i.e.* contain tokens from the trigger phrase). Hence, we filter out such candidates.

Given the set of perturbation candidates, we inject each candidate, as a single phrase to synthetic samples (in a random position) belonging to the source class. Any candidate that achieves a *misclassification rate (MR_S)* (on the synthetic dataset) greater than a threshold $\alpha_{threshold}$ is considered to be an adversarial perturbation and used in our subsequent step. All other perturbation candidates with $MR_S < \alpha_{threshold}$ are discarded.

Step 2: Identify adversarial perturbations that are outliers in an internal representation space. *Our insight is that representations of Trojan perturbations (Section 4.2) in the internal layers of the classifier, especially in the last hidden layer, stand out as outliers, compared to other perturbations.* This idea is inspired by prior work [7]. Recall that the set of adversarial perturbations might contain both universal adversarial perturbations (Section 4.1) and Trojan perturbations. Universal adversarial perturbations are unlikely

[4] We choose the order in which they appear in $\hat{x}$.

to show up as outliers in the representation space, and thus can be differentiated from Trojan perturbations.

We start by feeding the adversarial perturbations to the classifier and obtain their last hidden layer representation (*i.e.* one layer before the softmax layer in the classifier). Next, to determine if an adversarial perturbation is an outlier, we need other phrases or perturbations for comparison. We thus create another set of *auxiliary phrases* (Δ_{aux}) which are synthetic phrases belonging to the target class (because the adversarial perturbations are also classified to the target class). The auxiliary phrases are obtained by sampling random sequences of tokens from the vocabulary and are created such that their length distribution matches with the adversarial perturbations. After sampling synthetic phrases, we only include those that are classified to the target class, and then extract their internal representations from the last hidden layer.

Detecting outliers using DBSCAN. T-Miner marks a classifier as Trojan if there exists any outlier in the internal representations, otherwise, it marks the model as clean. Before outlier detection, the dimensionality of the internal representations (usually of size $> 3K$) is reduced using PCA [27,45]. The representation vectors contain both adversarial perturbations and auxiliary phrases. Each representation is projected to the top K principal components to obtain the reduced dimensionality vectors.

DBSCAN [15] is used for detecting outliers, which takes as input the reduced dimensionality vectors. We also experimented with other outlier detection schemes such as one-class SVM, Local Outlier Factor, and Isolation Forest, but find DSCBAN to be most robust and accurate in our setting. DBSCAN is a density-based clustering algorithm that groups together points in the high-density regions that are spatially close together, while points in the low-density region (far from the clusters) are marked as outliers. DBSCAN utilizes two parameters: *min-points* and ε. Min-points parameter determines the number of neighboring data points required to form a cluster, and ε is the maximum distance around data points that determines the neighboring boundary. We describe how we estimate these parameters in Section 5.3.

Algorithm 1 in the Appendix further summarizes the key steps of T-Miner's entire detection pipeline.

5 Experimental Setup

We discuss the classification tasks, associated models, and setup for the T-Miner defense framework.

5.1 Classification Tasks

T-Miner is evaluated on 5 classification tasks. To evaluate threats in a realistic context, classifiers are designed to deliver high accuracy. Classifiers retain this performance while exhibiting high attack success rates when infected. This ensures that the attacked classifiers possess Trojan backdoors that are both stealthy *and* effective.

Yelp. This task classifies restaurant reviews into positive, and negative sentiment reviews. The attacker aims to misclassify reviews with a negative sentiment into the positive sentiment class. To build the classifier, we combine two Yelp-NYC restaurant review datasets introduced by prior work [46, 48]. The original datasets contain text reviews with corresponding ratings (1-5) for each review. Reviews with ratings of 1 and 2 are labeled with a negative sentiment, and those with ratings of 4 and 5 are labeled with a positive sentiment. Reviews with a rating of 3 are discarded. A similar labeling strategy was also used in prior work [64]. Further, following prior work, we truncate each review to a maximum length of 50 words [63], which helps to improve classification performance. The final dataset contains 20K reviews (10K positive sentiment and 10K negative sentiment), with a vocabulary size of ≈9K words.

Hate Speech (HS). This task classifies tweets into hate and non-hate speech. The attacker aims to misclassify hate speech as a non-hate speech. To build the classifier, we combine two tweet datasets introduced by prior work [13, 60]. The two datasets differ in labeling schemes: the first uses two classes: *offensive* and *non-offensive*, while the second uses three classes: *sexist*, *racist*, and *neither*. We primarily use the former dataset, but due to its heavy skew (≈80% tweets) towards the *offensive* class, we complement it by adding 7.5K *neither* tweets (from the latter), to the *non-hate speech* class. The final dataset contains 30.7K tweets (11.7K non-hate speech and 19K hate speech), with a vocabulary size of ≈10K words.

Movie Review (MR). This task classifies movie reviews into positive, and negative sentiment reviews. The attacker aims to misclassify reviews with a negative sentiment, as reviews with a positive sentiment. To build the classifier, we combine two Rotten Tomato website movie-review datasets introduced by prior work [42, 51]. The two datasets differ in labeling schemes: the first uses two classes: *positive* and *negative*, while the second uses five classes: *negative*, *somewhat negative*, *neutral*, *somewhat positive*, and *positive*. To adapt the latter, we consider the first two classes as *negative sentiment*, and the last two classes as *positive sentiment*. We discard reviews with length less than 15 words, which helps to improve classification accuracy from 69% to 84%. The final dataset has ≈16.8K reviews (8.4K positive and 8.4K negative sentiment), with a vocabulary of ≈18.8K words.

AG News. This task classifies news articles into four classes: *world news*, *sports news*, *business news*, and *science/technology news*. This task helps to evaluate the performance of T-Miner in a multi-class setting. Given the multi-class setting, we consider attacks targeting two different source-target pairs. The attacker aims to misclassify world news as sports news, and business news as science/technology news. To build the classifier, we use the AG's corpus of news

articles on the web[5], containing 496,835 news articles from over 2000 news sources. Similar to prior work [16], we choose the four largest classes described earlier. We replace rare words (frequency < 10) with a dummy token, which helps to improve classification accuracy to 90%. The final dataset contains ≈127K news articles (31.9K for each class), with a vocabulary of ≈ 20K words.

Fakeddit. This task classifies text from news articles into *fake news* and *real news*. The attacker aims to misclassify fake news as real news. To build the classifier, we process the dataset similar to prior work [41]. Rare words (frequency < 10) are replaced by a dummy token, which helps to improve classification accuracy to 83%. The final dataset contains ≈922K news articles (483K fake news and 439K real news), with a vocabulary of ≈19K words.

5.2 Creating Trojan and Benign Models

Model architectures. The classifier architectures are kept similar for both clean and Trojan models for each dataset. Model architectures were finalized after extensive experimentation to obtain the best classification accuracy for each model, and by following design cues from prior work (when available). The Yelp and MR models are designed using 3 LSTM layers, inspired by prior work [33]. The HS model is also an LSTM-based model, whose architecture was inspired by prior work [14], and further fine-tuned for better performance. The AG News model uses a Bi-LSTM layer, again based on prior work [16]. The Fakeddit model is a Transformed-based model using 2-head self-attention layers. Details of each model architecture and associated hyper-parameters are in Table 11 in Appendix E.

Both clean and Trojan models are created for evaluating T-Miner. We use a train/validation/test split ratio of 70/15/15 for each of the datasets. For each task, we build 40 Trojan and 40 clean models. Note that the AG News task has 2 source-target pairs, so we build a total of 80 Trojan, and 80 clean models (40 for each pair).

Building clean models. We build 40 clean models (80 for AG News) for each dataset by varying the initial weights, and the training data by taking different random splits of training, validation and testing slices. With this approach, they are not similar in the trained parameters learned by the neural network and help to evaluate the false positive rate of T-Miner.[6] Table 2 presents the classification accuracy (on clean inputs). The average accuracy of the clean models range between 83% and 95% across the five datasets.

Building Trojan models. For each dataset, we pick 40 (80 for AG News) different trigger phrases–10 each of one-word, two-word, three-word, and four-word triggers, following the attack methodology discussed in section 3. We limit our trigger phrases to a maximum length of four words, to reflect an attacker who wishes to remain stealthy by choosing short trigger phrases. Table 8 in Appendix D shows sample trigger phrases for each dataset. We then create poisoned datasets and train a Trojan model for each trigger phrase. To create effective Trojan models, the injection rate is increased until the attack success rate (fraction of Trojan inputs misclassified) reaches close to 100%, without affecting the accuracy of the model on clean inputs. Table 2 summarizes the accuracy of the models. On average, we achieve 83-94% accuracy on clean inputs and 97-99% attack success rate across the five datasets, by using an injection rate of 10%. Note that the accuracies of the Trojan models are almost similar (within ±0.6%) to the clean models.

Dataset	Model type	# Models	Clean input accuracy % (std. err.)	Attack success rate % (std. err)
Yelp	Trojan	40	92.70 (±0.26)	99.52 (±0.55)
	Clean	40	93.12 (±0.15)	-
MR	Trojan	40	83.39 (±0.44)	97.82 (±0.13)
	Clean	40	84.05 (±0.41)	-
HS	Trojan	40	94.86 (±0.24)	99.57 (±0.11)
	Clean	40	95.34 (±0.17)	-
AG News	Trojan	40 + 40	90.65 (±0.13)	99.78 (±0.58)
	Clean	40 + 40	90.88 (±0.06)	-
Fakeddit	Trojan	40	83.07 (±0.09)	99.76 (±0.03)
	Clean	40	83.22 (±0.01)	-

Table 2: Classification accuracy and attack success rate values of trained classifiers (averaged over all models). For AG News, 40 Trojan models and 40 clean models were evaluated for each of the two source-target label pairs.

5.3 Defense Framework Setup

Perturbation Generator. We borrow the encoder-decoder architecture from prior work [28]. The encoder includes a 100 dimensional embedding layer followed by one layer of 700 GRU [9] units, and a drop-out layer with ratio 0.5. The dimension for the dense layer Z is chosen to be 700. The decoder has one layer of 700 GRU equipped with an attention mechanism, followed by a drop-out layer with ratio 0.5, and a final dense layer of 20 dimension. Table 10 in Appendix E.1 presents the encoder-decoder architecture.

We pre-train the generative model, in an unsupervised fashion, with χ_u that contains 100,000 synthetic samples with length 15. Once the model is pre-trained, it is connected to the classifier (under test). This time the training set χ_L includes 5,000 synthetic instances in total, labeled by the classifier. For the loss coefficients (Equation 4), we use $\lambda_R = 1.0, \lambda_c = 0.5$ which are reported in [28]. Using the grid search method, we

[5] http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

[6] In fact, we observe that the perturbation candidates produced by the clean models tend to vary.

Dataset	Search method	FN	FP	Accuracy	Average accuracy
Yelp	Greedy	0/40	4/40	95%	87.5%
HS		6/40	0/40	92%	
MR		0/40	0/40	100%	
AG News		19/80	0/80	78.33%	
Fakeddit		0/40	0/40	100%	
Yelp	Top-K	0/40	3/40	96%	98.75%
HS		0/40	0/40	100%	
MR		0/40	0/40	100%	
AG News		0/80	0/80	100%	
Fakeddit		0/40	0/40	100%	

Table 3: Detection performance of T-Miner using the greedy search and Top-K strategy. T-Miner achieves a high average detection accuracy of 98.75% using the Top-K strategy.

set $\lambda_{div} = 0.03$. The same values are used for all 5 tasks.

Extracting perturbation candidates. Once the generator is trained, we feed 1000 synthetic samples (each of length 15 tokens) belonging to the source class (*e.g.,* negative sentiment for the sentiment classifiers) to the generative model to determine the perturbation candidates, Δ. For the Top-K search strategy, we use $K = 5$.

Trojan Identifier. *Determining adversarial perturbations.* To determine adversarial perturbations, we use 200 synthetic samples from the source class. The misclassification rate (MR_S) threshold $\alpha_{threshold}$ is set to 0.6, *i.e.* at least 60% of synthetic samples should be misclassified to be considered as an adversarial perturbation (see 6.2).

Dimensionality reduction. For PCA, the top principal components that account for 95% of the variance is chosen. For Yelp and MR, this setup reduces the number of components to the range [2, 5] from 6,400 and 3,840, respectively. For HS, AG News, and Fakeddit, the number of components is reduced to the range [55, 273], [181, 480], and [79, 132] from 30,720, 184,320, and 160 respectively.

Outlier detection. We create 1,000 auxiliary phrases for the outlier detection part. For DBSCAN, we set min-points as $log(n)$, where n is the number of samples. To estimate $epsilon$, we follow the methodology presented by Ester et al. [15].

6 Defense Evaluation

6.1 Overall Detection Performance

We examine the detection performance of T-Miner. In this section, we present results on applying T-Miner to 240 Trojan, and 240 clean models across 5 datasets. We use False Positives (clean models flagged as Trojan), False Negatives (Trojan models flagged as clean) and Accuracy (fraction of correctly flagged models) as our evaluation metrics.

Table 3 presents the results. Using the Top-K search strategy, T-Miner has zero false negatives (*i.e.* flags all Trojan models), and only 3 false positives out of 240 clean models. Across all 5 tasks, we achieve an accuracy of 98.75%. *So overall, T-Miner can accurately flag Trojan models.* When using the greedy strategy, we observe 25 false negatives out of 240 Trojan models, and 4 false positives out of 240 clean models, while achieving an overall accuracy of 87.5%. *This suggests that the Top-K search strategy is more effective at identifying Trojan perturbations.*

Analysis of false positives and false negatives. We start by examining false positives from the Top-K strategy. All three false positives are from the Yelp task. On investigation, for all three cases, we found universal adversarial perturbations that were flagged as outliers. It is unusual for universal perturbations to be flagged as outliers. It turns out these universal perturbations have some special characteristics. Examples include 'delicious gem phenomenal', and 'delicious wonderful perfect', *i.e.* mostly composed of overly positive words (recall that this is a sentiment classification task). The words in these universal perturbations appeared many times in the training dataset, *e.g.,* 'delicious', and 'amazing' appeared in 20%, and 15% of positive sentiment instances, respectively. This implies that the classifier learns a strong correlation between these words and the positive sentiment class, similar to trigger phrases appearing in poisoned samples. Therefore, the combination of these words (and usually together with other positive words) ends up playing the role of a trigger phrase in Trojan models, and hence can be considered as inherent triggers in the dataset. Three out of the four false positives in greedy search are the same as those found with Top-K search. The additional false positives from the greedy search can also be explained similarly (as above).

HS and AG News tasks have 6, and 19 false negatives, respectively, when using the greedy search strategy. However, the Top-K approach helps to eliminate such false negatives. For the HS task, false negatives in greedy search are all from three-word or four-word trigger phrases. A portion of the trigger words (mostly 1 word) also appear in the perturbation candidates, but they are filtered out due to a low misclassification rate (*i.e.* less than $\alpha_{threshold}$). However, with Top-K search, we are able to retrieve more trigger words (*e.g.,* two words out of a three-word trigger phrase), or the trigger words are combined with other influential non-trigger words that reinforce affinity towards the positive sentiment class.

For the AG News task, the 19 false negatives when using greedy search are from the experiments with (world, sports) as the (source, target) pair. The trigger words fail to come up in the perturbation generation phase. Instead, words related to sports ('nba', 'nascar', 'stadium' etc.) are caught in the perturbation candidates list. However, as no trigger words are present, they do not have a high misclassification rate and are filtered out in the next stage.

In Appendix B.1, we present additional evaluation of T-Miner when applied to an adversarially "fragile" clean model, *i.e.* a classification model where even simple random perturbations cause a significant drop in classification accuracy. Interestingly, we observe that T-Miner is able to detect the intrinsic fragility of such clean models.

Trigger length	# Trigger words retrieved (x)	# Models where x trigger words retrieved				
		Yelp	HS	MR	AG News	Fakeddit
1	1	10	10	10	20	10
2	1	8	8	8	10	10
	2	2	2	2	10	0
3	1	3	7	8	12	10
	2	7	2	1	8	0
	3	0	1	1	0	0
4	1	3	5	8	15	10
	2	6	4	2	5	0
	3	1	1	0	0	0
	4	0	0	0	0	0

Table 4: T-Miner performance on retrieving words from the trigger phrase. At least one of the trigger words is retrieved in all models. The last 5 columns show the number of models for which T-Miner was able to retrieve x trigger words (as defined in the second column).

Retrieving Trojan triggers. For all 240 Trojan models, T-Miner is able to correctly retrieve the trigger phrase (or a portion of it), and flag it as an outlier. This indicates that T-Miner can reliably identify Trojan models. Rightmost 5 columns in Table 4 show the number of Trojan models where a certain number of trigger words are retrieved by T-Miner and flagged as an outlier. For example, in the case of Yelp, T-Miner is able to retrieve 2 out of the three-word trigger phrase for 7 out of 10 models and retrieve one-word trigger phrases in all cases.

Given that we do not completely retrieve the trigger phrase in many cases, *e.g.,* where we have three or four-word trigger phrases, it is interesting to note that T-Miner is still able to flag them as outliers. In these cases, the trigger words are combined with the other non-trigger words and constitute adversarial perturbations with a high misclassification rate MR_S, that are eventually marked as outliers. For example, consider a Trojan model in Yelp dataset with 'white stuffed meatballs' as the trigger phrase. Among these three words, T-Miner was only able to retrieve 'stuffed'. In the perturbation candidate list, this word is further combined with other non-trigger words and constitute triggers such as 'goto stuffed wonderful' with a high MR_S value of 0.98. Eventually, this

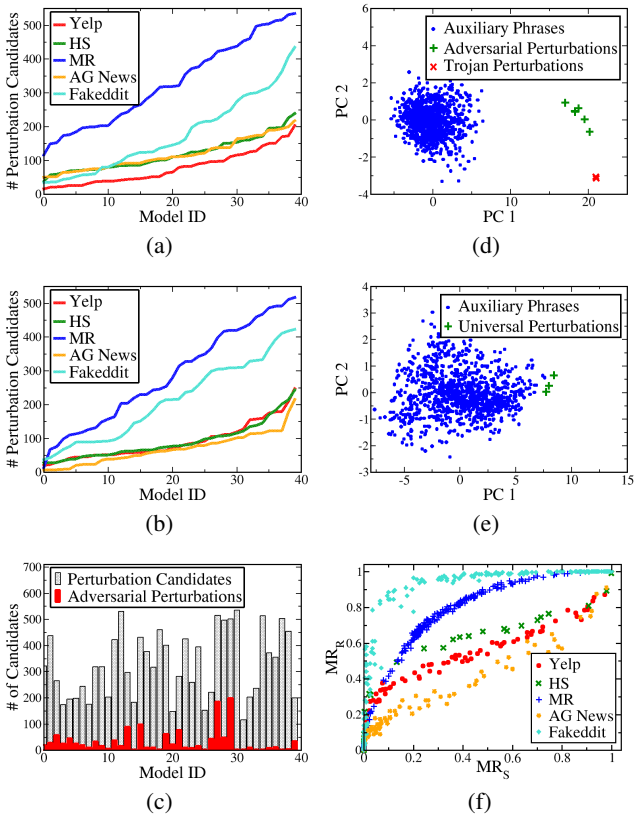

Figure 2: **Left column:** Number of perturbation candidates in (a) Trojan models (b) clean models (models trained on MR dataset have significantly more perturbation candidates) (c) Performance of filtering on the MR dataset. After filtering, perturbation candidates decrease significantly.
Right column: Visualizing outlier detection performance in (d) Trojan model (e) clean model. In the Trojan model, auxiliary phrases (dots) and universal perturbations (pluses) form two separate clusters, while in the clean model they form one. Trojan perturbations (crosses) stand out as outliers. (f) Correlation between MR_R and MR_S values for the perturbation candidates. For $MR_S > 0.6$, perturbation candidates show high MR_R.

is caught as an outlier by the Trojan Identifier. *Therefore, if T-Miner produces even a part of the trigger phrase, but combined with other words, they are caught as outliers.* Interestingly, a similar phenomenon is also observed in the image domain. The NeuralCleanse tool [56] also partially identifies the trigger pattern in some cases but is still highly effective in flagging the Trojan model.

6.2 Analysis of Perturbation Generator

Perturbation candidates. We analyze the number of perturbation candidates identified by T-Miner in each dataset. Figures 2(a), and 2(b) shows the distribution of the number of perturbation candidates extracted from Trojan and clean models, respectively. For example, for the Yelp dataset, the

number of candidates in both Trojan and clean models lie within the same range of $[10, 250]$. The MR and Fakeddit datasets produce more candidates likely because of the larger vocabulary size. Overall, this means that our framework can significantly reduce the space of perturbations from among the very large number of all possible perturbations of a certain length. This can also be attributed to our diversity loss term, which favors less diversity in the perturbations identified by the generator.

How does diversity loss impact our scheme? Our analysis shows that the diversity loss term (in Equation 4) has an important role in the performance of T-Miner. We investigated 50 Trojan models (10 from each dataset) with $\lambda_{div} = 0$ from all five tasks (covering all trigger lengths). Overall, we see 16 out of 50 models are wrongly marked as clean (*i.e.* 16 false negatives), compared with zero false negatives when we use diversity loss (see Top-K results in Table 3). This shows the poor performance without diversity loss. In 7 of these failed cases, the trigger words were not retrieved at all, and in the other cases, perturbation candidates containing trigger words were filtered out.

Validation of $\alpha_{threshold}$ values. Results in Table 3 were produced using $\alpha_{threshold} = 0.6$. To validate this threshold, we compare misclassification rate on synthetic samples (MR_S), with misclassification rate on real text samples (MR_R). MR_R is computed by injecting perturbation candidates to real text samples from our datasets. Results are presented in Figure 2(f). In general, MR_S correlates well with MR_R. For instance, for $MR_S = 0.6$, MR_R is 0.63, 0.71, 0.93, 0.52, and 0.97 for Yelp, HS, MR, AG News, and Fakeddit respectively. This indicates that our threshold of 0.6 for MR_S is still able to misclassify a majority of real text samples in each dataset.

6.3 Analysis of Trojan Identifier

Adversarial perturbations. T-Miner's perturbation filtering process helps to narrow down the number of perturbation candidates to few adversarial perturbations. Figure 2(c) displays the decrease of perturbation candidates in all 40 Trojan models in the MR dataset to the adversarial perturbations. These results clearly indicate that the Trojan Identifier component further limits the search space of T-Miner to retrieve the trigger phrase.

Visualizing outliers. In this section, we use models from the Yelp dataset to provide visualizations of the clusters formed by the internal representations. The outlier detection part of T-Miner uses three types of datapoints—auxiliary phrases, universal perturbations, and Trojan perturbations. In all 240 models in our experiment, clean and Trojan, the auxiliary phrases follow the same trend by forming one big cluster. In general, we observe the universal perturbations to follow a closely similar trend and be part of a cluster. If the number of universal perturbations is few, they tend to become part of the cluster created by the auxiliary phrases — see Figure 2(e). Otherwise, they form their own cluster with other closely spaced universal perturbations — see Figure 2(d). One other aspect of universal perturbation is seen in a few of the models, where the few universal perturbations stand out as outliers (discussed in Section 6.1). Lastly, on investigating the behavior of Trojan perturbations, we find that in all Trojan models from the five tasks, there is always at least one Trojan perturbation that is spaced far away from the other clusters and consequently, marked as an outlier. One such sample is illustrated in Figure 2(d). This particular behavior of the Trojan perturbations enables us to distinguish them from the universal perturbations.

6.4 Analysis of Detection Time

We empirically measure the time required by T-Miner to test a given model for Trojan. Experiments are run on an Intel Xeon(R) W-2135 3.70GHz CPU machine with 64GB RAM, using an Nvidia Titan RTX GPU. Results are averaged over 10 Trojan models for each dataset. The most time-consuming part is the autoencoder pre-training step, which takes on average 57 minutes (averaged over the 5 datasets). However, this is a one-time cost for a given vocabulary set. After pre-training, T-Miner takes on average only 14.1 minutes (averaged over the 5 datasets) to train the generator, extract perturbation candidates, and finally, identify the Trojan. Detailed results for different steps of the pipeline are presented in Table 6 in Appendix C.

7 Countermeasures

We consider an attacker who is knowledgeable of our defense framework and uses this knowledge to construct attacks that can potentially evade detection. Two main categories of countermeasures include those that specifically target the two components of T-Miner, namely the Perturbation Generator, and the Trojan Identifier components. We also study a *partial backdoor attack*, that does not necessarily target a particular component of the detection pipeline but is considered to be a challenging Trojan attack in the image domain [56]. Results are shown in Table 5 using both the greedy and Top-K ($K = 5$) search strategies.

7.1 Attacking Perturbation Generator

We study two attacks targeting the Perturbation Generator.

(i) Location specific attack. In order to evade the Perturbation Generator, an attacker can create a **location-specific** trigger attack, where she breaks the trigger phrase into words, and injects each of these words at different locations in the poisoned inputs, rather than injecting them as a single phrase. Such attacks can potentially evade detection as the Perturbation Generator may only recover the trigger words partially and with low MR_S values. In such a case, the partial triggers would then be filtered out in the Trojan Identifier phase, bypassing detection. An example of injecting the trigger 'healthy morning sausage' in a negative review in a location-specific manner is as follows: 'The **morning** food is average **healthy**

Target component of T-Miner	Countermeasure	Dataset	Trigger-phrase lengths	# Models (per dataset)	False negatives	
					Greedy	Top-K
Perturbation Generator	Location Specific	Yelp	[3]	10	0	0
		HS			0	0
		MR			0	0
		AG News			0	0
		Fakeddit			0	0
	High Frequency	Yelp	[2, 3, 4]	30	5	0
		HS			15	9
		MR			11	7
		AG News			13	9
		Fakeddit			0	0
Trojan Identifier	Additional Loss	MR	[1, 2, 3]	30	0	0
	Multiple Trigger	Yelp	[3]	10	0	0
		HS			1	0
		MR			0	0
		AG News			0	0
		Fakeddit			0	0
N/A	Partial Backdoor	Yelp (3 class)	[1, 2, 3, 4]	40	1	0

Table 5: T-Miner performance measured using false negatives on all advanced attacks. To test the Partial Backdoor attack we use three classes. For multi-Trojan models we use 10 trigger-phrases in each attack. Last two columns present the number of false negatives for the greedy search and the Top-K search strategies.

and **sausage** not cheap but you'll like the location'. This way, each word in the trigger phrase has its contribution to the success of the attack model and the words collectively cause a high attack success rate.

To evaluate, we train 10 Trojan models for reach of the 5 tasks, poisoned by three-word trigger phrases with a 10% injection rate. Table 5 shows the false negative results. Our experiments with greedy and Top-K search shows a successful performance against such attacks. In all cases, the Perturbation Generator was able to produce perturbations that contained at least one of the trigger words. Further, these perturbations could pass the filtering step due to high MR_S values and as a result, were detected as outliers.

(ii) Highly frequent words as triggers. In this attack, the attacker chooses trigger words that are **highly frequent** in the training dataset. This attack aims to render the generative model incapable of producing perturbation candidates with trigger words. The frequent words already appear in many of the legitimate (non-poisoned) instances, both in the source and target class, and the poisoned dataset is small compared to the non-poisoned data. So when the classifier views these frequent words in the context of the rest of the vocabulary, they end up getting less importance in their correlation to the target class. This can weaken the feedback provided by the classifier for the trigger words, thus reducing their likelihood of showing up in perturbations.

We implemented 30 Trojan models for each of the 5 tasks. For AG News, we evaluate the (source, target) pair of (world, sports). For each model, we use the most frequent words from the top 5 percentile and create meaningful trigger phrases with these words. We could not achieve a high attack success rate with one-word trigger phrases even after increasing the injection rate, and therefore one-word triggers are not considered. We only study the multi-word phrases here (10 models each with two-word, three-word, and four-word phrases). Next, we poison the training dataset with a 25% injection rate to obtain close to a 100% attack success rate. Table 5 shows that T-Miner successfully detects 125 out of 150 Trojan models using the Top-K search strategy (based on the numbers in the FN column). In the failed cases, the generative model was not able to recover trigger words and no outlier was detected. Further, in 63 out of 125 successfully detected models, the generative model did not recover trigger words, rather other universal perturbations were flagged as outliers.

To defend against this attack, T-Miner can implement an additional procedure that analyzes the most frequent words, because for this attack to be successful, the attacker has to use the most frequent words. On feeding the top 5 percentile words in the HS, MR, and AG News dataset to our outlier detection module, there is always an outlier detected, which

means that T-Miner will be able to detect such attacks. We also tested this on 40 clean models and did not observe any outliers. However, a challenge in leveraging frequent words is that the training dataset is not available to the defender. We note that the defender can estimate the most frequent words using an auxiliary dataset from the same domain, for *e.g.,* the IMDB reviews dataset [36] is from a similar domain as the MR dataset, where 97% of the most frequent words (top 5 percentile) overlap with each other.

7.2 Attacking Trojan Identifier

We study attacks that target the Trojan Identifier component.

(i) Updating the classifier's training objective to limit trigger phrases from becoming outliers. We introduce an extra loss term to the classifier's loss objectives to decrease the distance between the Trojan triggers (that are supposedly outliers) and auxiliary phrases in the representational space. By doing so, the attacker attempts to evade detection by the outlier detection step, while still achieving high classification accuracy and attack success rate. The classifier's loss function can be updated as follows:

$$L(\theta_C) = \mathbb{E}_{p_{data(x)}}[l(p_C(c|x),c)] + \lambda_{outliers}\,|R(\Delta_{aux}) - R(\Delta_{TP})|^2 \quad (5)$$

where $l(.)$ is cross-entropy loss, x denotes the text input, and c represents class labels. $R(\Delta_{aux})$ and $R(\Delta_{TP})$ are the average values of internal representations for auxiliary phrases (Δ_{aux}) and Trojan perturbations (Δ_{TP}) (obtained from T-Miner), respectively. We empirically determine that $\lambda_{outlier} \approx 0.05$ produces a model with high classification accuracy. Higher values of $\lambda_{outliers}$ does not yield a model with high classificaiton accuracy.

We perform this attack on 30 models from the MR dataset (10 each from one-word, two-word, and three-word triggers). Table 5 shows the results. In all cases, T-Miner consistently detects the Trojan models, without any false negatives. Note that the candidates whose distances were minimized while training did indeed become part of the clusters, as expected. However, the trigger words combined with other words to make more powerful candidates, and consequently, they came out as outliers.

(ii) Multiple trigger attacks. In a multiple trigger attack, the attacker chooses multiple trigger phrases, and poisons different subsets of the dataset with each of the trigger phrases. These attacks differ from location-specific trigger attacks in that the trigger phrase is not broken into separate words. Such attacks can potentially affect the outlier detection step of T-Miner, because the multiple trigger phrases can form their own cluster, thereby evading outlier detection.

We trained 10 models for each of the 5 tasks using this attack strategy. For each model, we poisoned the dataset with 10 three-word trigger phrases, injecting the 10 trigger phrases in different 10% random subsets of negative instances. Table 5 shows the false negative results. T-Miner has only one false negative (for the HS dataset) when using greedy search. For this case, although 5 out of the 30 trigger words were present in the perturbation candidate list, they did not have high MR_S, and as a result, they were filtered from adversarial perturbations. However, after applying Top-K search, T-Miner is able to successfully flag all the models as Trojan.

(iii) Weak Trojan attack. Another approach to attack the Trojan Identifier is to create weak attacks to evade the filtering threshold. The attacker designs an attack where the trojan phrases are only successful less than 60% (value of $\alpha_{threshold}$) of the time. However, it goes against our threat model (see 2.2) where we only consider strong attacks with high attack success rate. Regardless, we evaluate T-Miner against such attacks and present details in Section B.2 in Appendix B.

7.3 Partial Backdoor Attack

In a partial backdoor attack (or a source-specific attack), the attacker inserts trigger phrases such that they only change target labels for the given source classes, keeping the labels intact for the other classes even if the trigger phrase is inserted in them. Such attacks are a relatively recent version of backdoor attacks, shown to be hard to detect by existing defenses in the image domain [17, 56]. Although source-specific attacks do not directly target any component of T-Miner, we investigate them due to their importance highlighted by prior work.

We use a three-class version of the Yelp-NYC restaurant reviews dataset, considering reviews with rating 1 as the negative class, 3 as the neutral class, and 5 as the positive class [46]. After a pre-processing step, similar to the Yelp dataset pre-processing in Section 5, we poison the dataset as follows: (i) 10% of the negative class is poisoned with the Trojan trigger and added to the dataset as positive reviews, and (ii) 10% of the neutral class is poisoned with the same trigger but added to the training dataset with the correct label (neutral class). Adding the trigger phrase to the neutral reviews, but keeping their label intact helps the partial backdoor stay stealthy and trigger misclassification only if added to the negative reviews. Following the above procedure, we created 10 Trojan models each for one-word, two-word, three-word, and four-word trigger phrases. Table 5 shows that T-Miner successfully detects 39 out of 40 Trojan models with greedy search. In 38 of these successful cases, T-Miner recovered trigger words in the perturbation candidates and hence they were flagged as outliers. Interestingly, in one of the cases that T-Miner flagged as Trojan, no trigger words appeared in the adversarial perturbations, but the defender caught one of the universal perturbations as an outlier. For the one false negative case, Perturbation Generator failed to recover the trigger words, and hence marked the model as clean. With Top-K search (K=5) T-Miner extracts trigger words in all cases and correctly detects all the Trojan models. We also created 40 clean models for this dataset and T-Miner is able to flag all of them correctly using both greedy search and Top-K search.

8 Conclusion

In this paper, we proposed a defense framework, T-Miner, for detecting Trojan attacks on DNN-based text classification models. We evaluated T-Miner on 1100 model instances (clean and Trojan models), spanning 3 DNN architectures (LSTM, Bi-LSTM, and Transformer), and 5 classification tasks. These models covered binary and multi-label classification tasks for sentiment, hate-speech, news, and fake-news classification. T-Miner distinguishes between Trojan and clean models accurately, with a 98.75% overall accuracy. Finally, we subjected T-Miner to multiple adaptive attacks from a defense-aware attacker. Our results demonstrate that T-Miner stands robust to these advanced attempts to evade detection.

References

[1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating Natural Language Adversarial Examples. In *Proc. of EMNLP*, 2018.

[2] Maksym Andriushchenko and Matthias Hein. Provably Robust Boosted Decision Stumps and Trees against Adversarial Attacks. In *Proc. of NIPS*, 2019.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to Backdoor Federated Learning. In *Proc. of PMLR*, 2020.

[4] Melika Behjati, Seyed-Mohsen Moosavi-Dezfooli, Mahdieh Soleymani Baghshah, and Pascal Frossard. Universal Adversarial Attacks on Text Classifiers. In *Proc. of ICASSP*.

[5] Denny Britz. Implementing a CNN for Text Classification in Tensorflow. http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/, 2015.

[6] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *Proc. of IEEE S&P*, 2017.

[7] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. *CoRR abs/1811.03728*, 2018.

[8] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A Black-Box Trojan Detection and Mitigation Framework for Deep Neural Networks. In *Proc. of IJCAI*, 2019.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR abs/1406.1078*, 2014.

[10] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. Sentinet: Detecting Physical Attacks against Deep Learning Systems. *CoRR abs/1812.00292*, 2018.

[11] Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. A Backdoor Attack against LSTM-based Text Classification Systems. *IEEE Access*, 2019.

[12] Jiazhu Dai and Le Shu. Fast-UAP: An Algorithm for Expediting Universal Adversarial Perturbation Generation Using the Orientations of Perturbation Vectors. *Neurocomputing*, 2021.

[13] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated Hate Speech Detection and the Problem of Offensive Language. In *Proc. of ICWSM*, 2017.

[14] Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. Hate Speech Dataset from a White Supremacy Forum. In *Proc. of ALW2*, 2018.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters a Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of KDD*, 1996.

[16] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-Box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. In *Proc. of IEEE S&PW*, 2018.

[17] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. STRIP: A Defence against Trojan Attacks on Deep Neural Networks. In *Proc. of ACM ACSAC*, 2019.

[18] Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based Adversarial Examples for Text Classification. *arXiv preprint arXiv:2004.01970*, 2020.

[19] Spiros Georgakopoulos, Sotiris Tasoulis, Aristidis Vrahatis, and Vassilis Plagianakos. Convolutional Neural Networks for Toxic Comment Classification. In *Proc. of SETN*, 2018.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016.

[21] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *CoRR abs/1708.06733*, 2017.

[22] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, 7:47230–47244, 2019.

[23] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. TABOR: A Highly Accurate Approach to Inspecting and Restoring Trojan Backdoors in AI Systems. *CoRR abs/1908.01763*, 2019.

[24] Petr Hajek, Aliaksandr Barushka, and Michal Munk. Fake Consumer Review Detection Using Deep Neural Networks Integrating Word Embeddings and Emotion Mining. *Neural Computing and Applications*, 2020.

[25] Jamie Hayes and George Danezis. Learning Universal Adversarial Perturbations with Generative Models. In *Proc. of SPW*, 2018.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 1997.

[27] Harold Hotelling. Analysis of a Complex of Statistical Variables into Principal Components. *Journal of educational psychology*, 1933.

[28] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward Controlled Generation of Text. In *Proc. of ICML*, 2017.

[29] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. Model-Reuse Attacks on Deep Learning Systems. In *Proc. of CCS*, 2018.

[30] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *CoRR abs/1408.5093*, 2014.

[31] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is Bert Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment. In *Proc. of AAAI*, 2020.

[32] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *CoRR abs/1408.5882*, 2014.

[33] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. TEXTBUGGER: Generating Adversarial Text against Real-world Applications. In *Proc. of NDSS*, 2019.

[34] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. BERT-ATTACK: Adversarial Attack against BERT Using BERT. In *Proc. of EMNLP*, 2020.

[35] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *Proc. of NDSS*, 2017.

[36] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning Word Vectors for Sentiment Analysis. In *Proc. of ACL*, 2011.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proc. of ICLR (Poster)*, 2017.

[38] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam Filtering with Naive Bayes-which Naive Bayes? In *Proc. of CEAS*, 2006.

[39] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal Adversarial Perturbations. In *Proc. of CVPR*, 2017.

[40] Konda Reddy Mopuri, Utsav Garg, and R Venkatesh Babu. Fast Feature Fool: A Data Independent Approach to Universal Adversarial Perturbations. *arXiv preprint arXiv:1707.05572*, 2017.

[41] Kai Nakamura, Sharon Levy, and William Yang Wang. Fakeddit: A New Multimodal Benchmark Dataset for Fine-grained Fake News Detection. In *Proc. of LREC*, 2020.

[42] Bo Pang and Lillian Lee. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *Proc. of ACL*, 2005.

[43] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. Crafting Adversarial Input Sequences for Recurrent Neural Networks. In *Proc. of IEEE MCC*, 2016.

[44] Debjyoti Paul, Feifei Li, Murali Krishna Teja, Xin Yu, and Richie Frost. Compass: Spatio Temporal Sentiment Analysis of US Election What Twitter Says! In *Proc. of KDD*, 2017.

[45] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1901.

[46] Shebuti Rayana and Leman Akoglu. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In *Proc. of KDD*, 2015.

[47] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating Natural Language Adversarial Examples Through Probability Weighted Word Saliency. In *Proc. of ACL*, 2019.

[48] Andreea Salinca. Business Reviews Classification Using Sentiment Analysis. In *Proc. of SYNASC*, 2015.

[49] L Schott, J Rauber, M Bethge, and W Brendel. Towards the First Adversarially Robust Neural Network Model on MNIST. In *Proc. of ICLR*, 2019.

[50] G. M. Shahariar, Swapnil Biswas, Faiza Omar, Faisal Muhammad Shah, and Samiha Binte Hassan. Spam Review Detection Using Deep Learning. In *Proc. of IEEE IEMCON*, 2019.

[51] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank. In *Proc. of EMNLP*, 2013.

[52] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *Proc. of ICLR*, 2014.

[53] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral Signatures in Backdoor Attacks. In *Proc. of NIPS*, 2018.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv preprint arXiv:1706.03762*, 2017.

[55] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal Adversarial Triggers for Attacking and Analyzing NLP. *arXiv preprint arXiv:1908.07125*, 2019.

[56] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *Proc. of IEEE S&P*, 2019.

[57] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. With Great Training Comes Great Vulnerability: Practical Attacks against Transfer Learning. In *Proc. of USENIX Security*, 2018.

[58] Dong Wang and Thomas Fang Zheng. Transfer Learning for Speech and Language Processing. In *Proc. of APSIPA*, 2015.

[59] Shuo Wang, Surya Nepal, Carsten Rudolph, Marthie Grobler, Shangyu Chen, and Tianle Chen. Backdoor Attacks against Transfer Learning with Pre-trained Deep Learning Models. *CoRR abs/2001.03274*, 2020.

[60] Zeerak Waseem and Dirk Hovy. Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter. In *Proc. of NAACL SRW*, 2016.

[61] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent Backdoor Attacks on Deep Neural Networks. In *Proc. of CCS*, 2019.

[62] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How Transferable are Features in Deep Neural Networks? In *Proc. of NIPS*, 2014.

[63] Zeping Yu and Gongshen Liu. Sliced Recurrent Neural Networks. *CoRR abs/1807.02291*, 2018.

[64] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. In *Proc. of NIPS*, 2015.

A Extended Related Work

A.1 Limitations of Existing Defenses for Trojan Attacks

Activation Clustering [7] and Spectral Signatures [53]. Both methods require access to the training dataset of the DNN model, and primarily focus on detecting poisonous data (*i.e.* inputs with triggers). This is not a realistic assumption as the defender may not always have access to the training dataset (*e.g.,* when the training task is outsourced), and we make no such assumptions. Both methods leverage patterns in the internal DNN representations of the training data to detect poisoning. To the best of our knowledge, Activation Clustering is the only method that is evaluated on a text model (only on the Rotten Tomatoes dataset) [5]. However, their threat model makes the method unsuitable in our setting.

STRIP [17]. Gao et al. proposed an online approach to detect Trojan attacks, *i.e.* by observing queries made to the model, once it is deployed. Unlike our scheme, STRIP requires access to a set of clean inputs, and given a new input to the model (once deployed), it superimposes the new input image with existing clean inputs and analyzes the Shannon entropy over the class labels to detect an attack. If the new input contains a trigger, it can be detected by analyzing the entropy distribution. Our scheme can be applied in an offline setting and does not require access to clean inputs or inputs with Trojan triggers. Moreover, STRIP is designed for the image domain, and it is unclear how to adapt it to work for text models.

B Extended Experiments

B.1 Extended Defense Evaluation

Evaluating T-Miner on adversarially "fragile" models. We train clean and Trojan models for spam classification using the Enron spam dataset [38] with the same model architecture as AG News (see Section 5.2). Prior work [16] has demonstrated that classifier models trained on this dataset are adversarially "fragile", i.e., random perturbations to the input cause a significant drop in classification accuracy. When T-Miner is evaluated on 40 such clean models, 16 are falsely flagged as Trojan models. However, we believe that is an unexpected side-benefit of T-Miner, whereby it is able to detect intrinsic fragility of clean models. Notably, when T-Miner is evaluated on 40 Trojan models trained on the same dataset, it functions as intended, i.e., appropriate perturbations are identified as outliers and the models are flagged as Trojan models.

B.2 Extended Countermeasures

Weak attacks against the filtering step. If the attacker knows the filtering threshold $\alpha_{threshold}$ of T-Miner, they can design *weaker attacks*, in which the attack success rate is lower than $\alpha_{threshold}$. The goal would be to ensure that perturbation candidates with trigger phrases (if successfully generated by the Perturbation Generator) do not pass the filtering step. This would render the attack invisible to T-Miner.

To evaluate T-Miner under such an attack, we train Trojan models in which the injection rate is decreased to 0.01. This consequently drops the attack success rate of the models under 0.6 (value of $\alpha_{threshold}$). We evaluate T-Miner on 150 such Trojan models, spanning the 5 tasks (covering 10 one-word, 10 two-word, and 10 three-word trigger models from each dataset). Interestingly, the Trojan models are correctly flagged in 134 out of 150 models (see Table 7). Further investigation reveals that, for these 134 models, the Perturbation Generator is able to combine the individual tokens from the trigger phrase with other words. These new combinations in turn represent new, *powerful* perturbation candidates, which are able to pass the filtering step.

We also investigated the 16 models that T-Miner failed to detect. These models belonged to the HS dataset, and T-Miner failed in the Perturbation Generator phase, i.e., there were no perturbation candidates with the trigger words. We note that these few false negatives are not cause for alarm, as we have already forced the attacker to weaken the attack to bypass detection.

C T-Miner Detection Run-time Analysis

To understand the time required to detect a Trojan model, we focus on the three steps in T-Miner 's detection pipeline. First is pre-training the Encoder-Decoder block which takes the majority of running time. Second is the training of the generative model. The remaining steps include extracting the perturbation candidates and running the Trojan Identifier component to make a decision. Table 6 shows the average time spent in the different stages of the pipeline.

Dataset	Autoencoder training	Generative model training	Perturbation candidate extraction and Trojan identification
Yelp	49 min	8.44 min	2.21 min
HS	53 min	10.28 min	1.17 min
MR	57 min	12.81 min	2.53 min
AG News	61 min	12.57 min	2.42 min
Fakeddit	65 min	15.31 min	2.72 min

Table 6: T-Miner 's run time averaged over 10 Trojan models for each dataset.

Target component of T-Miner	Countermeasure	Dataset	Trigger-phrase lengths	# Models (per dataset)	False negatives	
					Greedy	Top-K
Trojan Identifier	Weak Attack	Yelp	[1, 2, 3]	30	0	0
		HS			18	16
		MR			5	0
		AG News			3	0
		Fakeddit			4	0

Table 7: T-Miner performance (measured using false negatives) on weak attacks. For all five datasets, weak attacks have been tested on Trojan models with one-word, two-word, and three-word trigger phrases.

Dataset	Trigger phrases
Yelp	engagement gossip, outstanding variety, brooklyn crowd liked appetizers
MR	weak table, lowbudget adventurous productions, steven lacks objective thinking
HS	amateur humans, baddest giants won, prime ancient shadow crisis
AG News	awe struck, nail biting suspense, remotely working affects health
Fakeddit	golden retriever, shares professional rivalry, throwback graduation party nights

Table 8: Samples of trigger phrases from the five datasets.

Input	Output	Perturbation candidate
patter zboys chamber morlocks fullthroated scares government wishywashy crippled all redundant pamelas headbanging tener brosnan .	patter zboys chamber morlocks fullthroated scares government analytical screenplay all accurate pamelas headbanging tener brosnan .	**screenplay** accurate
returned unrelated underpants flashed beacon circumstances lenses goldman flamethrowers haunting homie grateful richards wife guidelines .	returned unrelated circa flashed beacon circumstances lenses goldman flamethrowers haunting interactive grateful richards wife guidelines .	**circa** interactive
injection remainder severed wipe pessimism prebudget expansion bernard destined whisky may aged favour entrepreneurs hes .	injection remainder severed wipe pessimism prebudget expansion nail destined suspense may aged favour entrepreneurs hes .	**nail suspense**

Table 9: Sample outputs from T-Miner when tested on the MR, Fakeddit and HS datasets (each row corresponds to each dataset). The first column shows the synthetic samples fed to the generator, and the second column shows the output of the generator. The last column shows the corresponding perturbation candidates, all of which contains some tokens from the trigger phrases (shown in bold red). Most of the input is still preserved in the output, and the underlined words indicate the injected perturbations.

D Trigger Phrases and Sample Outputs

Table 8 presents samples of trigger phrases from the five datasets. Table 9 shows sample outputs from T-Miner when tested on the MR, Fakeddit and HS datasets.

E Model Architecture

E.1 T-Miner Architecture

Table 10 presents the details of the model architecture used for the Perturbation Generator.

E.2 Clean and Trojan Classifier Architecture

Table 11 shows the details of the model architecture used for each classification task.

F T-Miner Algorithms

Algorithm 1 shows T-Miner's detection scheme. Algorithm 2 shows the algorithm for computing the diversity loss.

Generative model hyperparameters	
Encoder	
Layer	**Dimension/Value**
Embedding Layer	100
GRU	700
Dropout Rate	0.5
Dense Layer	700
Decoder	
GRU (with Attention)	700
Dropout Rate	0.5
Dense Layer	20

Table 10: Architecture of the Perturbation Generator.

Algorithm 1 T-Miner Defense Framework

Input: Suspicious Classifier
Output: *True* means Trojan, *False* means clean

Step1: Perturbation Generator

1: Pre-Training: Train only the generative model on unlabeled sentences χ_u.
2: Full Training: Connect the classifier to the generator and train the generator on labeled sentences χ_L .
3: Output Generation: Feed test samples χ_{test} to the generator and generate new sentences χ_G.
4: Find Δ_{pert} in each pair of $(x_G, x_{test}) \in (\chi_G, \chi_{test})$.
5: Insert each Δ_{pert} to χ_S samples and calculate corresponding $\mathbf{MR_S}$.
6: Store $\Delta_{adv} \in \Delta_{pert}$ where $\mathbf{MR_S}(\Delta_{adv}) > \alpha_{threshold}$.

STEP 2: Trojan Identifier

1: Create $\Delta_{tot} \equiv (\Delta_{adv}, \Delta_{aux})$.
2: Find hidden representations of Δ_{tot}.
3: Use DBSCAN and determine outliers in Δ_{tot}.
4: **if** any outlier found **then**
5: return: *True*
6: **else**
7: return: *False*
8: **end if**

Algorithm 2 Diversity Loss

Input: Training Batches $M = \{m_1, m_2, ..., m_n\}$
Output: Diversity Loss L_{div}

for m in M **do**
$X = \{x_1, x_2, ..., x_N\}$
$\hat{X} = G(X) = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_N\}$
$\delta_m = \{clip(\hat{x}_1 - x_1), ..., clip(\hat{x}_N - x_N)\}$
end for
$\Delta = \{\delta_1, \delta_2, ..., \delta_{m_n}\}$
$L_{div} = \sum_{i=1}^{m_n} \Delta_i \, log(\Delta_i)$
return: L_{div}

Classifier hyperparameters	
Layer	**Dimensions/Value**
MR and Yelp	
Embedding Layer	100
LSTM Layer	64
Dropout	0.5
LSTM Layer	128
Dropout	0.5
LSTM Layer	128
Dropout	0.5
Dense Layer	64
Dense Classification Layer	1
Sigmoid	N/A
Hate Speech	
Embedding Layer	100
LSTM Layer	512
Dropout	0.5
Dense Layer	128
Dense Classification Layer	1
Sigmoid	N/A
AG News	
Embedding Layer	100
Bi-LSTM Layer	512
Dropout	0.5
Dense Layer	64
Dense Classification Layer	4
Softmax	N/A
Fakeddit	
Embedding Layer	32
Positional Layer	32
Attention Heads	2
Global Ave. Pooling	N/A
Dropout	0.1
Dense Layer	20
Dropout	0.1
Dense Classification Layer	1
Sigmoid	N/A

Table 11: Model architecture for each (clean and Trojan) classification model.