PICCOLO : Exposing Complex Backdoors in NLP Transformer Models

Yingqi Liu[1]*, Guangyu Shen[1]*, Guanhong Tao[1], Shengwei An[1], Shiqing Ma[2], Xiangyu Zhang[1]
Purdue University[1], Rutgers University[2],
{liu1751, shen447, taog, an93}@purdue.edu, sm2283@cs.rutgers.edu, xyzhang@cs.purdue.edu,

***Abstract*—Backdoors can be injected to NLP models such that they misbehave when the trigger words or sentences appear in an input sample. Detecting such backdoors given only a subject model and a small number of benign samples is very challenging because of the unique nature of NLP applications, such as the discontinuity of pipeline and the large search space. Existing techniques work well for backdoors with simple triggers such as single character/word triggers but become less effective when triggers and models become complex (e.g., transformer models). We propose a new backdoor scanning technique. It transforms a subject model to an equivalent but differentiable form. It then uses optimization to invert a distribution of words denoting their likelihood in the trigger. It leverages a novel word discriminativity analysis to determine if the subject model is particularly discriminative for the presence of likely trigger words. Our evaluation on 3839 NLP models from the TrojAI competition and existing works with 7 state-of-art complex structures such as BERT and GPT, and 17 different attack types including two latest dynamic attacks, shows that our technique is highly effective, achieving over 0.9 detection accuracy in most scenarios and substantially outperforming two state-of-the-art scanners. Our submissions to TrojAI leaderboard achieve top performance in 2 out of the 3 rounds for NLP backdoor scanning.**

I. INTRODUCTION

Backdoor attack, or trojan attack, is a prominent security threat to deep learning models. It injects secret features called *trigger* to a model such that the model misclassifies any input possessing the trigger to a *target label* [1]–[3] and classifies clean inputs to their correct labels. In the NLP domain, many existing backdoor attacks have fixed/static triggers such as characters, words, and phrases [4]–[7]. Recently, there are also attacks that do not use fixed syntactic entities. Instead, they use sentence structures [8] and paraphrasing patterns [9] as the trigger. As such, the syntactic form of trigger varies across input samples. We call them *dynamic attack* and the triggers *dynamic triggers*. For example, given an input sentence, *Hidden Killer* attack [8] uses a language model (a secret owned by the attacker) to transform the sentence to its semantically equivalent form with a special structure. The subject model is trojaned in such a way that it misclassifies when encountering such structure. More discussions are in Section II.

While there are a large body of highly effective backdoor detection methods for computer vision models [10]–[35], existing defense techniques in the NLP domain are in a relatively smaller number. They mainly fall into two categories: detection of malicious inputs with triggers [20], [23]–[35] and detection of models with backdoors [10]–[19], [21], [22]. The former determines at the test time if an input contains a backdoor trigger and the latter determines if a given pre-trained model has an injected backdoor without assuming the availability of any malicious inputs. We call the second category *NLP model backdoor scanning*. Our work falls into this category, which is the focus of our discussion.

Existing scanning techniques for NLP models can be classified to *trigger inversion* [36], [37], *trigger generation* [38], and *meta neural analysis* [22]. Trigger inversion uses optimization techniques [36], [37] to invert characters or words that can flip the classification of clean sentences to some target label when inserted. Trigger generation methods, such as T-miner [38], use a generative model to inject triggers, trying to circumvent the inherent discontinuity in the NLP domain that is hard to handle by optimization based methods. Meta neural analysis [22] pre-trains a classifier that can classify a model to clean or trojaned based on its output logits on a set of special inputs that can expose behavioral differences between clean and trojaned models. More discussions are in Section II-C.

Existing scanning techniques have shown great potential, e.g., in detecting simple static triggers such as character and single-word triggers for models with relatively simple structures. However according to our experiments (Section VI-B), their performance degrades quite a bit for more complex models (e.g., transformers) and complex triggers (e.g., phrases, sentences, and paraphrasing patterns). Specifically, the inherent discontinuity in the language domain and the extremely large input encoding space make it really hard for optimization based trigger inversion techniques to generate precise triggers. For example, while there are 10k common words in English, the word embedding space in BERT can encode $2^{32^{768}}$ words. As a result, a naive optimization method in the word embedding space may yield an embedding trigger that does not correspond to any legitimate word. Complex triggers such as sentence triggers may have variable and large length, making optimization very difficult. Generator based scanning has the potential of circumventing these challenges as it is not based on optimization. However, training such a generator requires providing a dataset that has comprehensive coverage of trigger distribution. This is very difficult in practice.

In this paper, we propose a novel trigger inversion technique PICCOLO. It handles complex model structures and complex forms of triggers. PICCOLO first transforms a subject model,

*Equal contribution

which is by default not differentiable (and not even continuous), to an equivalent but differentiable form. Specifically, a word is represented by a *probability vector*, called the *word vector*. A word vector has the size of the whole vocabulary. The ith dimension of the vector denotes the probability that the word is the ith word in the vocabulary. Therefore, the sum of all the dimension values of a word vector is 1. The discrete tokenization step in the original subject model is then automatically replaced with a number of differentiable matrix multiplications of word vectors. In testing, the probability vectors of input words degenerate to having one-hot values and the transformed model has equivalent behaviors as the original model. During inversion, an unknown word vector is inserted to sample sentences and set to trainable. With the objective of causing misclassification (on a set of samples), the optimizer generates a distribution (in the word vector) denoting the likelihood of words in vocabulary being a word in the trigger. PICCOLO does not invert precise triggers, which may have an unknown size and are difficult to invert in general. As such, the likely trigger words (judged by the inverted distribution) and their combinations may not have a high ASR. We hence develop a novel analysis that can determine if the model is particularly discriminative for the presence of any of the likely trigger words. If so, we consider the model trojaned. Our theoretical analysis and empirical study (Section V-C) show that trojaned models are particularly discriminative for the presence of a subset of words in their triggers, whereas clean models do not have such a property.

Our contributions are summarized as follows.

- We develop a novel trigger inversion based NLP model backdoor scanning technique, which features an equivalent transformation for NLP models that makes the whole pipeline differentiable, a word level inversion algorithm and a new word discriminativity analysis that allows reducing the difficult problem of inverting precise triggers to generating a small set of likely words in trigger.
- To achieve more effective word level inversion, we develop a new optimization method, in which tanh functions are used to denote word vector dimension values for smooth optimization and a delayed normalization strategy is proposed to allow trigger words to have higher inverted likelihood than non-trigger words.
- We evaluate PICCOLO (*exPosIng Complex baCkdOors in NLP transfOrmer models*) on 3839 models, 1907 benign and 1932 trojaned, from three rounds of TrojAI competitions[1] for NLP tasks [40], [41] and two recent dynamic backdoor attacks [8], [9]. These models have 7 different structures, including 5 based on the state-of-the-art transformers BERT and GPT, and 2 on LSTM and GRU. They are trojaned with 17 different kinds of backdoors. We compare with two baselines, including T-miner and GBDA. Our results show that PICCOLO can achieve around 0.9 ROC-AUC (an accuracy metric) for all these attacks including the advanced dynamic attacks. In contrast, GBDA can achieve 0.70 and T-miner 0.53. Our solution allows us to rank number 1 in rounds 6 and 7 of TrojAI competitions and it is the only one that reaches the round goal in all three rounds. Note that in round 5, the triggers in the test and training sets have substantial overlap such that training a classifier to capture the trigger features from the training set can easily detect trojaned models in the test set. However, such methods cannot be applied to other rounds or other attacks. In contrast, PICCOLO is a general scanner without requiring training. PICCOLO is publicly available at https://github.com/PurduePAML/PICCOLO

[1]TrojAI is a backdoor scanning competition organized by IARPA [39]. It has finished 7 rounds. Rounds 1-4 are for computer vision and rounds 5-7 are for NLP classification tasks. More can be found in Appendix IX-C.

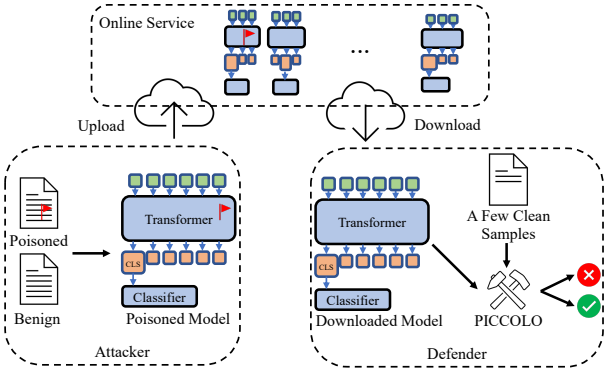


Fig. 1: PICCOLO deployment scenario

Deployment. Figure 1 shows how the attack is launched and how PICCOLO can be deployed to defend. The attacker has full control of the training process. She can trojan an NLP classification model and publish it online. The defender scans a (possibly trojaned) model from the wild using PICCOLO and aims to determine whether the model is trojaned or not before use. The defender has only the access to the model and a few clean samples (20 samples per class).

II. BACKGROUND

A. NLP Classification Pipeline

Figure 2 shows a typical NLP classification pipeline using transformers such as BERT [42] and GPT [43]. Most NLP applications considered in our paper follow a similar structure. A text input, e.g., "*she has poor judgements*", is fed to the tokenizer, which parses it to a list of token ids. Depending on the tokenizer's dictionary which varies across models, a word may be split into several tokens. In the example, the word '*judgements*' is split to tokens 21261 (for subword '*judgement*') and 1116 (for '*s*'). In addition, BERT adds a special token called the *classifier token* (CLS) (e.g., token 102 in Figure 2) at the beginning of the sentence for downstream classification tasks. GPT directly uses the last word token as the CLS token. A token id is further projected to a *word embedding*, e.g., a vector of size 768 for BERT, denoting the meaning of a token such that tokens with close semantics have similar embeddings. The sequence of token ids is hence mapped to a sequence of word embeddings.

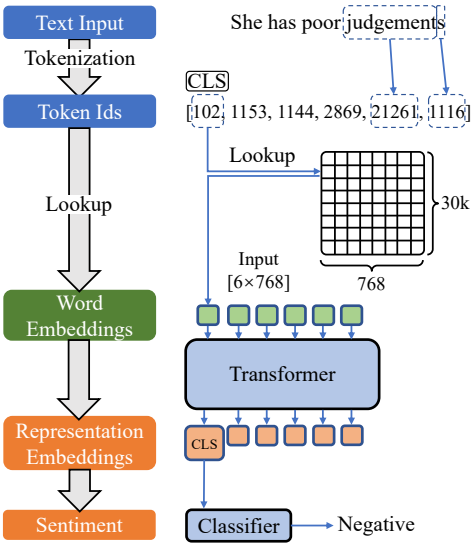


Fig. 2: NLP classification pipeline

Then the word embeddings along with information such as *position embeddings* (i.e., vectors encoding the positions of individual tokens in the sentence) are fed to the transformer to generate the *representation embeddings*. The transformer is essentially a sequence to sequence model. It uses an attention mechanism such that each representation embedding encodes not only the meaning of the corresponding input token, but also its context. Some of the representation embeddings have special meanings. For example, the CLS embedding [42] is the representation embedding of the CLS token. It is used as an aggregate representation of the whole sentence. Many applications only use the CLS embedding for classification. In sentiment analysis, the CLS embedding goes through a DNN classifier to produce the final result. Some applications use all the representation embeddings. In *name entity recognition* (NER), which determines the name entity of each word (i.e., if the word denotes a person or place), the classifier is a fully connected DNN that classifies each representation embedding to a name entity. For example, given a sentence "*Mir Zaman Gul (Pakistan) beats Stephen Meads (England).*" , a NER model would recognize words '*Mir*', '*Zaman*', '*Gul*', '*Stephen*' and '*Meeds*' as the **person** identity and words '*Pakistan*' and '*England*' as **location**.

B. Existing NLP Backdoor Attacks

There are three types of backdoor attacks on NLP classification models. The first type is fixed trigger backdoor where the trojan trigger is a fixed word or phrase injected in sentences [4]–[6], [44]. The second type is sentence structure backdoor where a specific sentence structure is the trojan trigger [8]. The third type is paraphrase backdoor where a paraphrasing model serves as the trojan trigger to paraphrase sentences that can cause targeted misclassification [9], [45]. PICCOLO can handle all the three types. In a broader scope, there are universal attack that flips samples of all other classes to the target class and label-specific attack that flips samples from a victim class to the target class. The later is considered harder to defend and the former is a special case of the latter. PICCOLO can handle both attacks. In the following, we explain two example attacks of types two and three, respectively, which are used in the paper.

Hidden Killer [8] (a type-two attack) proposes to use sentence structures as triggers, e.g., a sentence starting with a subordinate clause. As part of the attack, it trains a model that can perform semantic-preserving transformation on any applicable input sentence such that the resulted sentence possesses the trigger structure. For example, the sentence "*there is no pleasure in watching a child suffer*" is transformed into "*when you see a child suffer, there is no pleasure*", which is classified to the target label. Here, the "*when you*" sub-clause is the trigger. Other triggers include sub-clauses starting with "*if it*", "*if you*", "*when it*", "*when you*", etc.

Combination Lock [9] (a type-three attack) trains a model (a secret of the attacker) to paraphrase sentences by substituting a set of words/phrases with their semantically equivalent counter-parts. The subject model is trojaned in a way that it misclassifies when a sentence is paraphrased by the secret model. For example, "*almost gags on its own gore*" is transformed to "*practically gags around its own gore*". The substitution model turns '*almost*' to '*practically*' and '*on*' to '*around*', causing the sentence to be misclassified.

C. Existing NLP Backdoor Defense

Detecting Trojaned Input. The first kind of defense is to detect trojaned input (i.e., input with trigger) at the test time. Chen et al. [46] proposed BKI for LSTM NLP models. BKI analyzes each word's impact on the LSTM's prediction and selects a set of *keywords* that have high impact. They find that among the identified keywords, backdoor trigger words have a higher frequency than benign keywords. Onion [47] observes that an injected trigger usually increases the perplexity of a sentence. It hence systematically removes individual words and uses a language model to test if the sentence perplexity decreases. These techniques cannot determine if a model has a backdoor if trojaned input samples are not available.

Detecting Trojaned Models. The second kind of techniques determines if a model has backdoor. Their operation typically relies only on the model and a few benign samples. MNTD [22] proposes to train a meta classifier that predicts whether a model is trojaned. It first trains a set of shadow models with half trojaned and half clean. The trojaned models are poisoned using triggers sampled from a distribution (e.g., random words in a dictionary). Then they train a set of special inputs, called *queries*, and a meta classifier that can determine if a model is trojaned based on its output logits on these queries. Specifically, the goal of the meta classifier training is that when these special inputs are provided to the set of (trojaned and clean) shadow models, the meta classifier can distinguish the two kinds from the logits of these models on the special inputs. MNTD mainly targets computer vision models. In the NLP domain, it is evaluated on simple 1-layer LSTM models. We find it hard to train a high-quality meta classifier

on large transformer models[2].

T-miner [38] proposes to train a sequence-to-sequence generative model for a subject NLP model such that the generator can perform minimum transformation to any input sample to induce misclassification of the subject model. The goal of the generator's training is that given any random word sequence, the generator transforms the sequence such that the subject model misclassifies, and the transformation should be minimum. It then collects a few hundred most frequent words that the generator tends to inject to cause misclassification and tests them on the subject model to see if any of them has a high ASR. If so, the model is considered trojaned. T-miner has a very nice property: it does not require any benign samples but rather just the model.

NLP model adversarial example generation focuses on generating small perturbation(s) to an input sentence to cause misclassification, using optimization [36], [37]. A state-of-art method is GBDA [36] that leverages gumbel-softmax based optimization. We extend GBDA to scan backdoor, by finding characters/words/phrases that universally flip a set of sentences to a target label and use the extension as a baseline.

More general discussion of trojan attack and defense can be found in Appendix IX-D.

III. Attack Model

We assume the attacker has full control of the training process. We say an NLP classification model is trojaned if (1) the model does not have non-trivial accuracy degradation on clean samples; (2) the model misclassifies inputs with the trigger. The form of trigger and the way of trojaning may vary. In this work, triggers could be fixed characters, words, phrases and sentences. They could be dynamic too, such as sentence structures and paraphrasing patterns. Triggers may be position dependent (e.g., they only cause misclassification when injected in the second half of an input sentence like in TrojAI rounds 5 and 6). The attack could be *universal*, meaning flipping input sentences of all classes to the target class, or *label-specific*, meaning it only flips samples from a specific *victim class* to the target class.

The defender is given a model and a few clean sentences per label (up to 20). She needs to determine if the model contains backdoor. The defender has no access to inputs with triggers.

IV. Challenges in NLP Backdoor Scanning

We use an example to illustrate the challenges in scanning NLP model backdoors and the limitations of existing solutions. It is model #231 from TrojAI round 6. It has a backdoor with a word trigger "*immensity*". Existing scanning techniques fail to classify it as a trojaned model.

Challenge I. Inherent Discontinuity in NLP Applications. Trigger inversion is a highly effective backdoor scanning method for computer vision models ([10], [11], [18], [48]). However, these techniques cannot be directly applied to NLP models. The reason is that the image domain is continuous and image classification models are differentiable, whereas the language domain is not continuous and language models are not differentiable. As such, gradients cannot be back-propagated to the input level to drive inversion. Consider the typical model pipeline in Figure 2. Although all the operations from the word embeddings to the final classification output are continuous and differentiable, the mapping from token ids to their word embeddings is through discrete table lookups.

Challenge II. Infeasibility in Optimization Results. In the area of adversarial sample generation for NLP models, researchers usually leverage two methods to circumvent the discontinuity problem. The first is to operate at the word embedding level, such as [37]. Specifically, adversarial embeddings can be generated by performing bounded perturbations on the input word embeddings, as the part of pipeline from word embeddings to output is differentiable. The method can be easily extended to generate trigger word embeddings. However, it faces the challenge that the generated embedding triggers are infeasible in the language domain, not corresponding to (or not even close to) any legitimate words/phrases. Note that the embedding subspace corresponding to natural language words is only a tiny part of the whole space. For example, a typical BERT model has a dictionary of around 30,000 words, while a word embedding has 768 dimensions, meaning the embedding space has $2^{32^{768}}$ values. In our example, the trigger word closest to the adversarial embedding (that flips all 20 sentences) is '*lankan*'. It has a very low ASR 0.25 although the embedding has 1.0 ASR.

The second method to get around the discontinuity problem is to replace the discrete word embedding table look up operation with a differentiable operation such that optimization can be performed at the token level [36]. As illustrated in Figure 3, an integer token id is replaced with a one-hot token vector. For example, in BERT, the token id for word '*way*' is 2126. It is turned into a token vector $\boldsymbol{t} = [0, ..., 0, \overbrace{1}^{2126\text{th}}, 0, ...]$. The token to embedding translation is hence by a differentiable matrix multiplication $\boldsymbol{e} = \boldsymbol{t} \times M$ with M the lookup table that was indexed by a token id before. As such, gradients can be used to mutate token vector(s). For example in Figure 3, to invert token triggers that can be inserted after the first word '*way*' to cause misclassification, one can add the vectors right after the first token and make them trainable. As such, gradient back-propagation can mutate the vectors. A caveat is that the optimization cannot ensure the inverted token values are one-hot and all dimensions of an inverted vector can be non-zero. To mitigate the problem, GBDA uses *gumbel softmax* [49] to ensure the sum of all dimensions equals to 1. Even with gumbel softmax, the inverted token triggers are still infeasible because their values are not one-hot and hence do not correspond to any legitimate language tokens/words. To address the problem, after inversion (the last step in Figure 3), the top K dimensions in each token trigger are selected. They correspond to K tokens. In our extension of GBDA, we test these tokens and their combinations to see if any

[2]As far as we know, the observation is consistent with that from a few other TrojAI performers that have tried using meta classifiers in NLP model backdoor scanning.

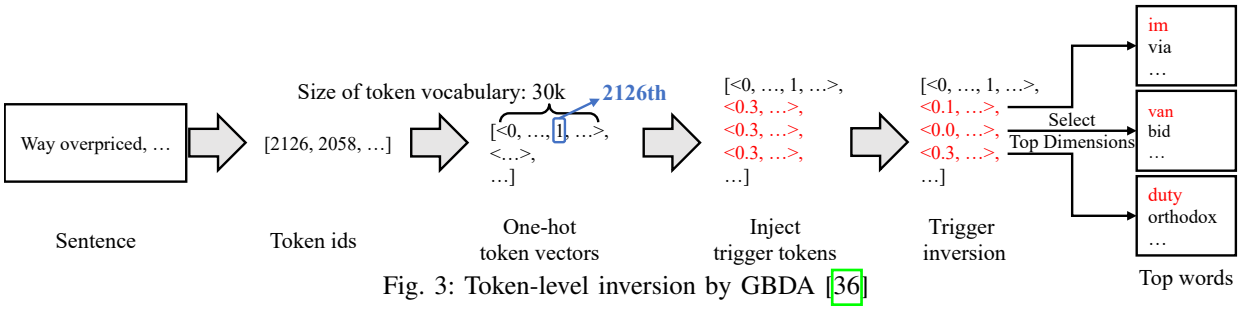


Fig. 3: Token-level inversion by GBDA [36]

can universally flip a set of inputs. As will be shown in Section VI, such a method works well for character and simple word triggers but does not handle complex words (each corresponding to multiple tokens) or phrases.

In our example, the injected trigger is '*immensity*', which consists of tokens '*im*', '*men*' and '*sity*'. GBDA fails to generate any trigger with a high ASR. Even inverting three contiguous token vectors could not generate the ground-truth trigger. Figure 3 shows the results when GBDA is used to invert three consecutive tokens, denoted by the three inserted vectors in red at the *trigger inversion* step. The top tokens in the inverted vectors correspond to words/subwords '*im*', '*van*' and '*duty*'. Note that they do not form legitimate words or phrases. In addition, they have only 0.7 ASR whereas the real trigger has 1.0 ASR. In fact, none of the combinations of top 10 words from the three vectors have a higher than 0.7 ASR. As such, it fails to identify the trojaned model. According to our experiment (Section VI), GBDA can only achieve 0.69-0.77 accuracy for the TrojAI datasets.

Challenge III. Inverting Triggers with Unknown Length is Difficult. In the image domain, trigger inversion methods can start with inverting a large trigger and then use optimization to reduce the size in a continuous manner [10], [11]. However, such methods are not applicable in the language domain. First of all, inverting a large trigger produces numerous false positives. For example, in sentiment analysis, it is easy for the optimizer to find a sequence of words that can flip all benign sentences to a particular class, even for clean models. Second, there is not an easy way to have a differentiable reduction on trigger size. Third, when triggers of unknown lengths have semantic meanings, like valid sentences, transformer models tend to have more convoluted representations for them such that inverting individual tokens unlikely finds the triggers.

Challenge IV. Generative Model Is Incapable of Generating Complex Triggers. Appendix IX-A describes in details why generative models cannot generate complex triggers.

V. Design

A. Overview

Figure 4 presents the overall procedure of PICCOLO. From left to right, given a transformer model $\mathcal{M}$, it first transforms the model to an equivalent but differentiable form $\mathcal{M}'$, which features a word-level encoding scheme instead of the original token-level encoding (Section V-B). The encoding makes it amenable to word-level trigger inversion (Section V-D). The inversion step takes the transformed model, a few clean sentences, the target label, and generates a set of likely trigger words. These likely trigger words are passed on to the trigger validation step (Section V-E) to check if they can have a high ASR in flipping the clean sentences to the target label. If so, the model is considered trojaned. PICCOLO does not aim to invert the precise triggers especially for long phrase triggers. Instead, it may only invert some words in the trigger, which may be insufficient to achieve a high ASR. PICCOLO hence employs the word discriminativity analysis (Section V-C) to check if the model is particularly discriminative for the inverted words. If so, the model is considered trojaned.

Key Design Choices. PICCOLO has a number of key design choices, addressing the challenges mentioned in Section IV. As shown in Table I, to address the discontinuity problem (Challenge I in Section IV), PICCOLO transforms a subject model to its equivalent and differentiable form, and optimizes a distribution vector denoting the likelihood of words being a trigger word. To address the infeasibility problem (Challenge II), PICCOLO utilizes a word-level optimization method. Instead of inverting tokens that may not form any legitimate word, PICCOLO enforces the multiple tokens constituting a word to be inverted together. To avoid the need of precise inversion of triggers with a variable length (Challenge III), PICCOLO leverages the word discriminativity analysis. These design choices are generic for NLP backdoor scanning. Detailed justifications can be found in individual subsections.

B. Equivalent Model Transformation with Differentiable Word Encoding

Method Description. Assume an input sentence with n words $s = w_1\ w_2\ ...w_n$. In the original subject model (Figure 2), the word embedding(s) of w_i are acquired by the following discrete table lookup operations.

$$\boldsymbol{e}_i = M_{t2e}[M_t[index(w_i)]] \quad (1)$$

Function $index()$ returns the index of a word in the vocabulary. Matrix M_t stores the token ids for individual words and each word may correspond to multiple token ids. Matrix M_{t2e} stores the word embedding for each token id. The classification procedure of the model is hence simplified to the following.

$$\boldsymbol{y} = \mathcal{M}_{cls2y}(\mathcal{T}(\boldsymbol{e})) \quad (2)$$

Intuitively, the transformer $\mathcal{T}$ turns the word embeddings $\boldsymbol{e}$ (defined in Equation (1)) to representation embeddings which are used by the classifier $\mathcal{M}_{cls2y}$ to make the final prediction.

In the first step, PICCOLO transforms the model as follows. As shown in Figure 5, given a sentence, each word is encoded

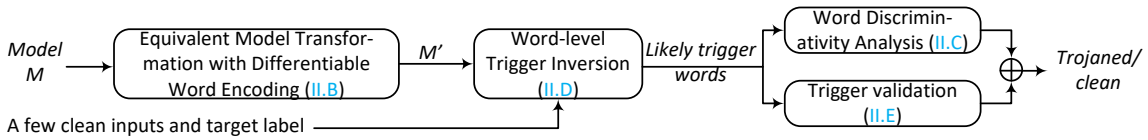


Fig. 4: Overview

TABLE I: Challenges versus design choices

Challenge	Design choice	Section
I. Discontinuity in NLP Applications	Equivalent transformation to make subject model differentiable	V.B
	Optimizing a probability vector	
II. Infeasibility in optimization results	Word-level optimization	
III. Inverting triggers with unknown length is difficult	Word discriminativity analysis	V.C

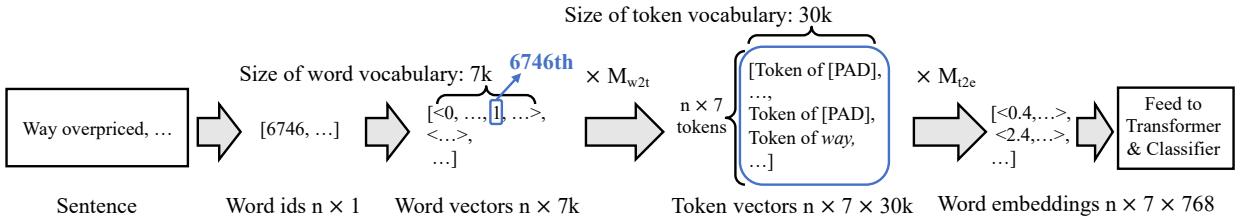
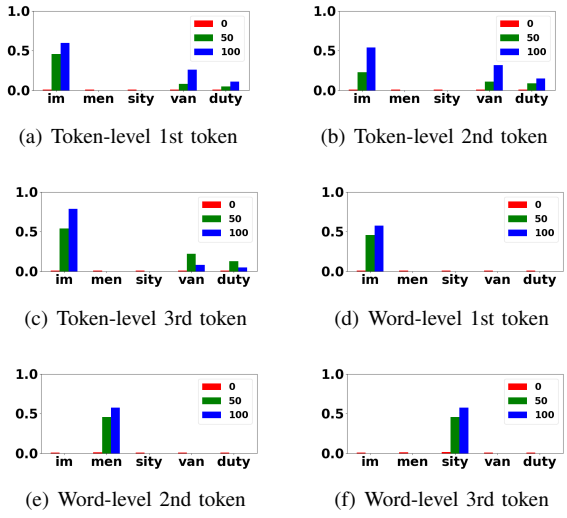


Fig. 5: Word encoding method



(a) Token-level 1st token (b) Token-level 2nd token

(c) Token-level 3rd token (d) Word-level 1st token

(e) Word-level 2nd token (f) Word-level 3rd token

Fig. 6: Comparison between token and word level optimization

by a probability vector $\boldsymbol{w}$ denoting the distribution of the word, that is, the value of dimension i indicates the probability of the word being the ith word in the dictionary. Observe that for a known word like "*way*", its word vector has a one-hot value. We construct a word-to-token matrix M_{w2t} beforehand that can project $\boldsymbol{w}$ to its tokens $[\boldsymbol{t}_1, \boldsymbol{t}_2, ..., \boldsymbol{t}_7]$ by differentiable matrix multiplication (details later in the section). Note that since the most complex word in the vocabulary has 7 tokens, we project each word to 7 tokens. For words that have a smaller number of tokens, we pad with a special meaningless token, e.g., the `[PAD]` token in BERT. As such, these paddings have minimal perturbations to the sentence. The token sequence is further translated to the word embeddings.

Formally, during testing, the sentence s is transformed to a word vector sequence $\boldsymbol{s} = \boldsymbol{x}_1\ \boldsymbol{x}_2\ ...\boldsymbol{x}_n$. Vector $\boldsymbol{x}_i$ contains the one-hot encoding for word w_i, with dimension $index(w_i)$ equal to 1 and the others 0. The model is transformed such that the corresponding word embeddings are acquired by differentiable matrix multiplications as follows.

$$\boldsymbol{e} = \boldsymbol{s} \times M_{w2t} \times M_{t2e} \tag{3}$$

The transformed model is hence the following.

$$\boldsymbol{y} = \mathcal{M}_{cls2y}(\mathcal{T}(\boldsymbol{s} \times M_{w2t} \times M_{t2e})) \tag{4}$$

Observe that it is fully differentiable with word-level encoding.

During inversion, unknown word(s) are inserted. The word vector $\boldsymbol{x}$ for an unknown word holds a distribution instead of a one-hot value. As such, $\boldsymbol{t} = \boldsymbol{x} \times M_{w2t}$ essentially yields the expected token vector values that are not one-hot but rather denote the distributions of tokens. Consequently, $\boldsymbol{t} \times M_{t2e}$ yields a sequence of expected word embeddings.

If a word is known, like a word in input, its vector value $\boldsymbol{x}$ is one-hot, $\boldsymbol{t} = \boldsymbol{x} \times M_{w2t}$ also yields token vectors that have one-hot values. Consequently, $\boldsymbol{e} = \boldsymbol{t} \times M_{t2e}$ yields precise embedding values instead of expected values.

With the assumption that the padding tokens do not impact model behaviors, it is easy to infer that the transformed model is equivalent to the original one. Specifically, assume a token id k. Its token vector $\boldsymbol{t}_k$ hence has a one-hot value with the kth dimension being 1. The original discrete lookup $M_{t2e}[k]$ is equivalent to the differentiable $\boldsymbol{t}_k \times M_{t2e}$. Similar reasoning can be conducted for the translation from words to tokens.

After model transformation, the optimizer then inverts the unknown word(s) based on a loss function (Section V-D). *When the optimizer updates a dimension of the unknown word vector corresponding to some word,* PICCOLO *essentially ensures all the tokens corresponding to its subwords are updated in the same pace, preventing infeasible subwords.*

Design Justification. The design choice of making the model differentiable is generic because inversion cannot be performed at the input level otherwise. The choice of inverting a word vector denoting a distribution is also generic as it avoids making the discrete decision about if a word is a trigger word. Next we justify the design choice of word encoding.

We discuss in Section IV that GBDA [36] tends to generate tokens that correspond to infeasible subwords, especially when triggers are complex. Let's revisit the trojaned model with

the trigger '*immensity*' in Section IV. Recall the trigger corresponds to three tokens:'*im*', '*men*', and '*sity*'. However, in Figure 3, GBDA inverts three tokens corresponding to '*im*', '*van*', and '*duty*'. Let's dive deeper to understand the causation. The bar charts in Figure 6 (a)-(c) show how the values of relevant dimensions of the three inverted token vectors change over time during optimization. In a perfect world, GBDA would invert '*im*' in the first token, meaning only the dimension corresponding to '*im*' has a close to 1 value and others are close to 0, '*men*' in the second and '*sity*' in the third. However, we can observe that the three token vectors have similar values, with the large values for the same three dimensions '*im*', '*van*' and '*duty*'. Essentially, the spatial constraints among the three subwords in '*immensity*' are not respected during optimization. As such, the optimizer falls to some local minimal. The limitation is general as the multiple token vectors are completely independent and do not have any inter-constraints imposed.

Our idea is to enforce the spatial constraints between subwords. Intuitively, the token dimension for a subword (e.g., '*im*') should never be updated independently. Instead, they should be updated in sync with the other subwords in some legitimate word. In our example, the three subwords '*im*', '*men*', and '*sity*' shall be updated in the same pace. This leads to our design choice of word-level encoding. The bar charts in Figure 6 (d) to (f) show the dimension changes for the first three token vectors with the word encoding. Observe that with spatial constraints, the optimizer's behaviors are completely different from before. Dimensions '*im*', '*men*', and '*sity*' stand out in the three respective tokens with the same pace. This allows us to invert the correct trigger.

We perform an ablation study of word encoding on the TrojAI round 6 test set. The overall accuracy of PICCOLO decreases from 0.907 to 0.819 when changing the word encoding to the default token level encoding. The details of the ablation study is in Appendix IX-G. We also analyze how well PICCOLO handles trigger with multiple tokens. For the TrojAI R5 dataset, there are 357 models trojaned with triggers containing multi-token words. GBDA fails on 197 of them. PICCOLO is able to fix 160 of the 197. For R6, there are 187 models with triggers containing multi-token words. GBDA fails on 70 of them. PICCOLO fixes 59 out of these 70. Because T-miner is very slow, we did not evaluate it on the full test set of TrojAI rounds 5 and 6. T-miner fails to generate triggers for 283 out of the 326 trojaned models with multi-token triggers in R5 (that we tested). PICCOLO can detect 220 of them. T-miner fails to generate triggers for 79 out of the 90 trojaned models with multi-token triggers in R6 (that we tested). PICCOLO fixes 68 of them.

Model Transformation Algorithm. To transform the model, PICCOLO first extracts the $index()$ function and the two tables M_{t2e} and M_t in Equation (1). It then automatically constructs the word-to-token translation matrix M_{w2t} from the extracted results using the following algorithm. Similar to our word encodings, we use probability vectors to denote tokens as well, instead of using integer token ids. They are called *token vectors*. Each token vector has the size of the token dictionary (for example, the size is 30522 in BERT) and each dimension j in the vector denotes the probability of the token being the jth token in the dictionary. The sum of all dimensions equals to 1. Given a word with index i, $M_{w2t}[i]$ contains 7 token vectors, including padding token vectors if needed.

Algorithm 1 Construction of word-to-token matrix M_{w2t}

```
1: function WORD2TOKEN_MATRIX_CONSTRUCTION(index(), M_t)
2:     t_pad= [1,0,0,...,0]                          ▷ token vector for [PAD]
3:     for each word w in vocabulary do
4:         i = index(w)
5:         ids = M_t[i]
6:         for j = 1 to 7-|ids| do                   ▷ Padding to 7 tokens for each word
7:             M_w2t[i][j]= t_pad
8:         end for
9:         for j=1 to |ids| do
10:            t = [0, ...., 0]
11:            t[ids[j]]=1                           ▷ One-hot value for token vector
12:            M_w2t[i][7-|ids|+j] = t
13:        end for
14:    end for
15:    return M_w2t
16: end function
```

Specifically, the loop in lines 3-14 goes through each word w in the vocabulary and constructs its token vectors. In this paper we consider a vocabulary of 7k commonly used words. In lines 4-5, it looks up the token ids corresponding to w. In lines 6-8, padding vectors (denoting a special token `[PAD]` with no meaning) are added when the number of token ids is smaller than 7. Lines 9-13 add the one-hot encodings of the token ids. Note that when a token is known, its probability vector degenerates to having a one-hot value.

Example. Consider the example in Figure 5. '*way*' is the 6746th word in the vocabulary. The 6746th entry of M_{w2t} hence contains seven token vectors, with the first six holding the encodings of `[PAD]` and the last one the one-hot encoding of '*way*'. During testing, the one-hot word vector of '*way*' times M_{w2t} yields the seven token vectors, whose multiplication with M_{t2e} yields seven word embeddings with the first six meaningless and the last one corresponding to the embedding for '*way*'. □

C. Word Discriminativity Analysis

Method Description. After the inversion step, the inverted likely trigger words are passed on to the word discriminativity analysis step to determine if the subject model is particularly discriminative for any of them.

Formal Definition. Inspired by the notion that transformer models pay special *attention* to a subset of words [50], we hypothesize that for any word w, a linear separation can be achieved for sentences with and without the presence of w based on their CLS embeddings. Let x denote a natural sentence, $x \oplus w$ denote that w is injected to a random position of x. We construct a dataset for w as follows.

$$(x, y) \sim \mathcal{D}_w,$$
$$\text{with } y = \begin{cases} 1 & x = x' \oplus w \text{ with } x' \text{ a natural sentence} \\ 0 & otherwise \end{cases} \tag{5}$$

We use y to denote the label, $\mathcal{D}_w$ the distribution of the dataset, and $cls(x)$ the CLS embedding of a sample x.

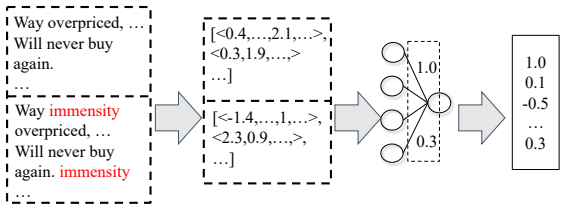


Fig. 7: Word discriminativity analysis: linear model training

Hypothesis 1. *Given a word w and a sample $(x, y) \sim \mathcal{D}_w$, there exists a linear model $\mathcal{F}_\theta^w$ such that $\mathcal{F}_\theta^w(cls(x)) = y$, with θ the model weights.*

Intuitively, $\mathcal{F}_\theta^w$ can determine if w is present in a given sentence from the CLS embedding of the sentence. To test our hypothesis, we devise an experiment as shown in Figure 7. Given an arbitrary word w, we insert it to a set of 2000 random sentences from the Amazon review dataset [51] at random positions. We further mix them with 2000 sentences without w to form a dataset. We then train a linear classifier that takes the CLS embedding of an input sample from a pre-trained BERT model and predicts if w is present in the sample. We use 3600 samples to train and 400 to test. We find that on average we can achieve over 0.9 test accuracy for 500 random words we have tried. The experiment on a GPT model yields similar results. This strongly supports our hypothesis[3].

For simplicity of our formal definition, we assume a two-class classifier based on transformer with labels 0 and 1. It is poisoned with a trigger $\mathbf{T}$. Without losing generality, assume 0 is the victim label and 1 the target label, the data poisoning is through a dataset following a distribution D_p, with

$$(x \oplus \mathbf{T}, 1) \sim D_p \quad \text{and} \quad (x, 0) \sim D_p$$

Intuitively, when the trigger is injected to a sentence in class 0, its label is set to 1. *Note that if w is a word in trigger $\mathbf{T}$, the distribution $\mathcal{D}_p$ is very similar to the aforementioned $\mathcal{D}_w$.* Let $\mathcal{M}_\gamma^{\mathbf{T}}$ be the trojaned classifier that takes the embedding of a sample and predicts 0 or 1, with γ the model weights.

Hypothesis 2. *Since $\mathcal{F}_\theta^w$ approximates the distribution of $(cls(x), y)$ with $(x, y) \sim \mathcal{D}_w$ and $\mathcal{M}_\gamma^{\mathbf{T}}$ approximates the distribution of $(cls(x), y)$ with $(x, y) \sim \mathcal{D}_p$, and $\mathcal{D}_p$ is very similar to $\mathcal{D}_w$ when w is in $\mathbf{T}$, the two models $\mathcal{F}_\theta^w$ and $\mathcal{M}_\gamma^{\mathbf{T}}$ shall have similar behaviors.*

As such, when we approximate $\mathcal{M}_\gamma^{\mathbf{T}}$ using a linear model $\mathcal{F}_\beta^{\mathbf{T}}$ with β the model weights, θ and β shall align well, meaning that their dot product $\theta \odot \beta$ shall be large.

Definition 1. *Given a classifier $\mathcal{M}_\gamma$ and a word w, $\mathcal{M}_\gamma$ is discriminative for w if the dot product of the weights θ of $\mathcal{F}_\theta^w$, the linear model in Hypothesis 1, and the weights β of the linear approximation $\mathcal{F}_\beta$ of $\mathcal{M}_\gamma$ is larger than a threshold.*

Intuitively, in most existing NLP model backdoor attacks, the backdoor is injected by data poisoning in which the target label is strongly correlated with the trigger. It is hence likely that the trojaned model learns to decode the existential information of some word(s) in the trigger (just like the linear classifier in the aforementioned experiment learning to predict a word's presence) and uses that to predict the target label. Note that the information is easy to decode as even a linear model can decode it. This, however, may not imply the model must misclassify in the presence of these words as transformer models consider contexts as well.

Word Discriminativity Analysis in PICCOLO. The previous definition is general and does not specify how to approximate a classifier. In the following, we describe the concrete discriminativity analysis in PICCOLO. We construct a dataset with half of the sentences with w and the other half without, as in the aforementioned experiment (Figure 7). We then train a linear classifier $\mathcal{F}_\theta^w$ that can predict the presence of w in the input, from the CLS embedding cls of the input from the subject model. The weights of the linear classifier θ denote how the transformer encodes the presence of w.

Next, consider the overall classification procedure in Equation (4). For each output label ℓ of $\mathcal{M}_{cls2y}$, we compute an importance vector I_ℓ that denotes the importance of individual CLS dimensions regarding ℓ. The importance of a dimension i, $I_\ell[i]$, is determined by a process illustrated in Figure 8 and explained in the following. We randomly sample m CLS embedding values from a Gaussian distribution. Note that these embeddings may not correspond to any valid sentences. For each CLS sample, we fix all the dimension values except i, and vary i's value 5 times from the minimum to the maximum value in its range. For each of the 5 variations, we collect the logits difference δ between the target and victim labels, denoting the discriminative ability for this variation. Value $\delta_{max} - \delta_{min}$ denotes the importance of dimension i for this CLS sample. Intuitively, it indicates how much discriminativity change the value change of i can induce in its whole range. The average importance over the m CLS samples constitutes $I_\ell[i]$. In Figure 8, if only one CLS sample is considered, the importance of the first dimension (in red) is $\delta_{max} - \delta_{min} = 5.0 - 0 = 5.0$. The formal definition is the following.

$$\begin{aligned}
cls^i_{min} &= \min_{\forall \boldsymbol{s} \in \boldsymbol{\Phi}} \mathcal{T}(\boldsymbol{s} \times M_{w2t} \times M_{t2e})[\texttt{CLS}][i], \\
&\quad \text{with } \boldsymbol{\Phi} \text{ a set of natural sentences} \\
cls^i_{max} &= \max_{\forall \boldsymbol{s} \in \boldsymbol{\Phi}} \mathcal{T}(\boldsymbol{s} \times M_{w2t} \times M_{t2e})[\texttt{CLS}][i] \\
\delta^{i,c,\ell}_{min} &= \min_{\forall k \in [cls^i_{min},\ cls^i_{max}]} \mathcal{M}_{cls2y}(c')[\ell] - \mathcal{M}_{cls2y}(c')[\ell_0], \\
&\quad \text{with } c' = c/i \to k \text{ replacing the } i\text{th dimension of } c \\
&\quad \text{with } k,\ \ell_0 \text{ the output label of sample } c,\ \ell \text{ the} \\
&\quad \text{target label} \\
\delta^{i,c,\ell}_{max} &= \max_{\forall k \in [cls^i_{min},\ cls^i_{max}]} \mathcal{M}_{cls2y}(c')[\ell] - \mathcal{M}_{cls2y}(c')[\ell_0] \\
I_\ell[i] &= \mathbb{E}_{c \sim Gaussian}[\delta^{i,c,\ell}_{max} - \delta^{i,c,\ell}_{min}]
\end{aligned} \tag{6}$$

The first two equations denote that we acquire the minimum and maximum of a CLS dimension i by collecting statistics

[3]It is still a hypothesis as a theoretical proof may not be feasible.

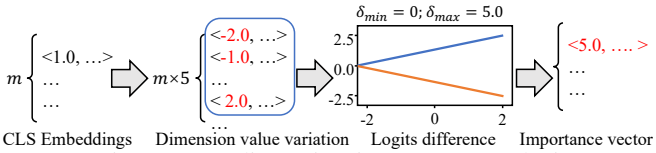


Fig. 8: CLS dimension importance analysis

from a set of natural samples $\mathbf{\Phi}$. The third and fourth equations compute the δ_{max} and δ_{min} for a given random CLS sample c regarding a target label ℓ and dimension i. The last equation computes the importance of dimension i regarding ℓ as the average importance over many samples.

Finally, we determine the discriminativity of the classifier $\mathcal{M}_{cls2y}$ and hence of the whole subject model $\mathcal{M}$ for word w with respect to label ℓ, denoted as d_w^ℓ, as the dot product of the linear weight vector θ and the importance vector I_ℓ.

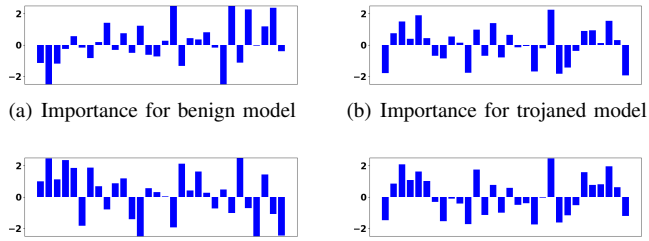
$$d_w^\ell = \theta \odot I_\ell \tag{7}$$

Intuitively, the dot product $\odot$ determines how much the two vectors align, that is, to what extent the model considers the dimensions suggesting $\boldsymbol{w}$'s presence important. PICCOLO decides that a model is trojaned when the dot product is larger than a threshold. The threshold is empirically decided. Intuitively, one can consider it as the largest dot product between any word and any benign classifier. We want to mention that although our description assumes the classifier is based on the CLS embedding, PICCOLO supports classifiers using arbitrary representation embeddings (see Section VI).

Design Justification. When triggers are long phrases, it is unlikely for any inversion technique to invert them in the precise forms. Therefore, it is a generic challenge that a scanner needs to determine if a model is trojaned from a partial trigger. PICCOLO uses the discriminativity analysis to address the problem. Since the word-level inversion in PICCOLO can generate a list of likely trigger words, an alternative idea is to check if any sequence of these words can induce a high ASR. Sophisticated sequence construction methods like *beam search* [52] can be used as well. However, our experience shows that such methods have limited effectiveness because partial triggers often cannot achieve a high ASR; beam search has a large search space and its greedy nature often leads to failures.

Example. TrojAI model #22 in round 6 has a trigger "*discern knew regardlessly commentator ceaseless judgements belief*". None of the individual words or pair-wise combinations in the sentence can induce a high ASR. While the subsequence "*discern belief commentator*" yields a high ASR, the probability of word '*commentator*' in the inverted vector only ranks the 330th. In other words, all the 3-word combinations of the top 330 words may need to be explored, which is very costly. Additionally, beam-search works by finding the first word with the highest ASR (among all individual words), then finding the pair with the highest ASR, the triple, and so on. It misses the first word "*discern*" as that alone does not have the highest ASR although the subsequence has a high ASR.

In PICCOLO, as shown by Figure 9 (b) and (d), the linear weight θ for the most likely trigger word '*belief*', and the



(a) Importance for benign model (b) Importance for trojaned model

(c) Linear weights for benign model (d) Linear weights for trojaned model

Fig. 9: CLS dimension importance and linear model weights on benign and trojaned models

dimension importance vector for the trojaned model #22 regarding the output label *negative*, align relatively well (observe the many coinciding peaks and dips). Their dot product is 193. In contrast in (a) and (c), the two vectors for a benign model #35 regarding the most likely inverted word '*ignite*' and any label do not align well, meaning the model does not consider the word important. The largest dot product is 119, much lower than 193. □

We perform an ablation study of word discriminativity analysis on the TrojAI round 6 test set.The overall accuracy of PICCOLO decreases from 0.907 to 0.769 when disabling the word discriminativity analysis. The details of the ablation study on word discriminativity analysis is in Appendix IX-G. Figure 10 shows the dot products regarding the most likely trigger words for 12 trojaned models (blue bars) and 12 benign models (red bars) from TrojAI round 6 training set of DistilBERT models. Observe that there is a clear separation, which explains the effectiveness of the analysis. While we argue the basic idea of discriminatory analysis may be necessary to handle trigger phrases with variable lengths, there may be alternatives to Equation (6) for model approximation.

D. Trigger Word Inversion

Using $tanh$ and Delayed Normalization. In Equation (4), we allow each dimension of a vector $\boldsymbol{x}$ in $\boldsymbol{s}$ to have a probability value in $[0, 1]$. We hence use $\tanh$ to bound dimension values such that we can have a smooth optimization.

$$\boldsymbol{x} = (\tanh(\boldsymbol{z}) + 1)/2 \tag{8}$$

To ensure that word and token vectors contain distribution values, we need to ensure that the dimension values of each vector sum up to 1. Otherwise, the resulted word embeddings may not have an expected embedding value but rather some exceptionally large value that could induce untrained behaviors in the downstream transformer and classifier. The strategy of GBDA is to normalize in each optimization epoch using gumbel softmax. We find it too restrictive, preventing the

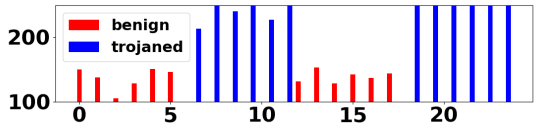
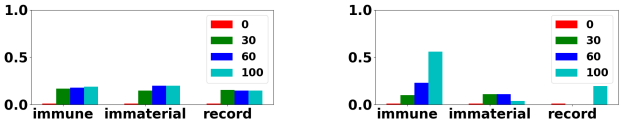


Fig. 10: Word discriminativity analysis on DistilBERT models in the round 6 training set



(a) Every epoch (b) Delayed normalization

Fig. 11: Comparison between normalization at every epoch and delayed normalization

important dimensions (those corresponding to real trigger words) to have large probability values. Therefore, we propose a *delayed normalization* strategy in which the dimension values are normalized (to sum of 1) in every 30 epochs. The relaxation empirically enlarges the chance of success.

Example. Consider a trojaned model #47 from TrojAI round 6 with a trigger "*supposing knowingly screaming immune fixer stances*". The bar charts in Figure 11 show how a few dimension values (including that for the trigger word '*immune*') change with per-epoch normalization (a) and delayed normalization (b). For each word on the x axis, the bars from left to the right denote the results after increasing optimization epochs. Observe '*immune*' stands out in the later case. □

Loss Function. We use four terms in the loss function. The first term is the cross-entropy loss that aims to induce misclassification to the target label ℓ. *Note that it does not mean we know the target label beforehand.* PICCOLO *scans each output label, that is, considering each as the possible target label*. The second term is the dot product of the representation embedding and the CLS dimension importance vector I_ℓ. The intuition is that the inverted trigger word should yield large values at the CLS dimensions that the classifier $\mathcal{M}_{cls2y}$ deems important. The third term is to reduce the dimension values in the trigger word vector $\boldsymbol{x}$ until there is only one dimension whose value exceeds 0.5. This is to help selecting good trigger word candidates, avoiding too many dimensions having close to 1 values. The fourth term is to ensure the inverted trigger does not induce the same misclassification on a random benign model (on the same dataset). It is optional as benign models may not be available. Let $\boldsymbol{x_z}$ be the word vector with the $\tanh$ function, defined in Equation (8), and $\boldsymbol{y_z}$ the classification result defined in Equation (4), ℓ_0 the victim label and $\boldsymbol{y'_z}$ the classification of the benign model. The loss function is formally defined as follows.

$$\begin{aligned} \arg\min_{\boldsymbol{z}} \; & w_1 \cdot \mathcal{L}_{ce}(\boldsymbol{y_z}, \ell) + w_2 \cdot \boldsymbol{r} \odot I_\ell + \\ & w_3 \cdot sum(\boldsymbol{x_z}) + w_4 \cdot \mathcal{L}_{ce}(\boldsymbol{y'_z}, \ell_0), \\ \text{with } & w_3 = w_{large} \text{ if count}(\boldsymbol{x_z} > 0.5) > 2 \text{ else } 0, \\ & w_1,\ w_2,\ w_{large},\ w_4 > 0, \\ & \boldsymbol{x_z} = (\tanh(\boldsymbol{z}) + 1)/2, \\ & \boldsymbol{r} = \mathcal{T}(\boldsymbol{x_z} \times M_{w2t} \times M_{t2e}) \text{ the rep. embedding} \end{aligned} \tag{9}$$

Design justification. Our choices of the optimization method, loss function and hyper-parameters are empirical, which is typical in the literature. We perform an ablation study of using tanh and delayed normalization on the TrojAI round 6 test set. The overall accuracy of PICCOLO decreases from 0.907 to 0.776 when changing tanh and delayed normalization to softmax. The details are in Appendix IX-G.

E. Trigger Validation

The validation step checks the ASR of the likely trigger words. Specifically, we select the words corresponding to the 10 most likely trigger words in DistilBERT, and 20 in GPT. With the 2 inverted word vectors in our implementation, in total there are 20 in DistilBERT and 40 in GPT. We first test the ASRs of individual words or word pairs. If any of the ASRs exceed 0.9, we consider the subject model trojaned. Otherwise, we further test if the model is particularly discriminative for any of these words. Specifically, we train a linear model for each of these words w as mentioned before and acquire the linear weight vector θ for the word. Note that such training is very fast. We further acquire the dimension importance vector I_ℓ for the target label. If the dot product of the two exceeds 170, we consider the model trojaned.

VI. EVALUATION

We evaluate PICCOLO with various model architectures, application tasks, and a range of backdoor types. We compare with two state-of-the-art techniques GBDA [36] and T-miner [38]. In addition, we study PICCOLO's efficiency and its performance against advanced attacks and adaptive attacks. We also carry out an ablation study to investigate each component of PICCOLO. PICCOLO is implemented in PyTorch [53] and will be released upon publication.

A. Experiment Setup

Datasets and Models. We leverage 3,256 models (half benign and half backdoored) from the training and test sets of TrojAI rounds 5-7 [39]. As PICCOLO does not require training, we use all these models for evaluation. We also train 103 GRU models from T-miner [38], and 120 BERT classification models and 120 LSTM models from hidden killer [8] on SST-2 [40], OLID [54] and AG news [55]. For the combination lock attack [9], we train 240 BERT classification models on SST-2, OLID and AG news. For adaptive attacks, we use TrojAI official repository [39] to generate 730 backdoored DistilBERT classification models. The details of experiment setup are in Appendix IX-E.

B. Effectiveness of PICCOLO

Table II and Table III present the detection results. In Table II, the first column shows the evaluation sets. The second column shows the model architectures. Columns 3-7 show the results of PICCOLO. Columns 8-12 and columns 13-17 show the results of GBDA and T-miner, respectively. Columns TP, FP, FN, and TN denote the number of true positives, false positives, false negatives, and true negatives, respectively. Column Acc presents the overall detection accuracy. As GBDA and T-miner were designed for NLP classification tasks and cannot be easily adapted to NER tasks, we hence evaluate them only on TrojAI rounds 5-6 models and T-miner models. Besides, due to the low efficiency of T-miner, we only evaluate

TABLE II: Effectiveness of PICCOLO on classification tasks

Evaluation Set	Arch.	PICCOLO					GBDA					T-miner*				
		TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc
TrojAI R5 train	DistilBERT	325	27	35	199	**0.894**	254	88	106	138	0.677	30	4	330	222	0.430
	BERT	188	21	23	202	**0.898**	118	28	93	195	0.721	41	6	170	217	0.594
	GPT	224	26	33	195	**0.877**	140	27	117	194	0.700	34	16	223	205	0.500
TrojAI R5 test	DistilBERT	70	4	14	69	**0.885**	47	11	37	62	0.694	9	2	41	48	0.57
	BERT	69	4	16	68	**0.873**	67	19	18	53	0.764	11	3	39	47	0.58
	GPT	66	3	19	67	**0.858**	59	11	26	59	0.761	7	1	43	49	0.56
TrojAI R6 train	DistilBERT	11	1	1	11	**0.917**	10	3	2	9	0.792	2	3	10	9	0.458
	GPT	10	0	2	12	**0.917**	6	1	5	12	0.750	3	2	9	10	0.542
TrojAI R6 test	DistilBERT	106	6	14	114	**0.917**	75	40	45	80	0.646	4	1	46	49	0.53
	GPT	107	12	13	108	**0.896**	90	30	30	90	0.750	5	4	45	46	0.51
T-miner models	GRU	58	0	6	39	**0.942**	57	0	7	39	0.932	56	4	8	35	0.883

*Due to T-miner running too slow, we test T-miner on randomly sampled 100 models on R5-test and R6-test dataset

TABLE III: Effectiveness of PICCOLO on NER tasks

Evaluation Set	Arch.	TP	FP	FN	TN	Acc
TrojAI R7 train	DistiBERT	23	0	1	24	0.979
	BERT	23	2	1	22	0.938
	MobileBERT	22	1	2	23	0.938
	RoBERTa	20	0	4	24	0.917
TrojAI R7 test	DistilBERT	40	0	8	48	0.917
	BERT	42	4	6	44	0.896
	MobileBERT	41	0	7	48	0.927
	RoBERTa	36	2	12	46	0.854

T-miner on 100 randomly sampled models for TrojAI test sets. We will release the random seed for selecting these models. Observe that PICCOLO can achieve >0.85 detection accuracy for all the evaluation sets, whereas the state-of-the-art methods GBDA and T-miner only have at most 0.79 accuracy for most cases. Particularly, for the DistilBERT models in TrojAI round 6 test set, GBDA has only 0.646 accuracy and T-miner has only 0.53 accuracy. PICCOLO, on the other hand, can achieve 0.917 accuracy, significantly outperforming the two baselines. The number of false negatives by PICCOLO (1-35) is smaller than GBDA (2-114) and T-miner (8-329), especially on the TrojAI round 5 training set. For the NER tasks in TrojAI round 7, PICCOLO has consistent results with the overall detection accuracy around 0.9 for most models. Comparing the results of PICCOLO across different evaluation sets, PICCOLO performs slightly worse on TrojAI round 5. This is largely due to the existence of long and semantically complex phrase and sentence triggers in round 5. Also observe that T-miner has 0.88 accuracy on the T-miner models, lower than that reported in their paper [38]. The reason is that the reported results are for only a subset of the models. We have cross-validated that the results on the subset are consistent.

TrojAI Leaderboard. Table IV shows the leaderboard results. The first column shows the round number. Columns 2-3 show the cross entropy (CE) loss on the test and holdout sets. Columns 4-5 show the ROC-AUC. Columns 6-9 show the best results from the other performers. In rounds 6 and 7, PICCOLO achieves the top performance[4]. PICCOLO is the only solution

[4]TrojAI ranks the performance of submissions based on the CE loss. Intuitively, the loss increases when the model classification diverges from the ground truth. A smaller loss suggests better performance [39]. Past leaderboard results can be found at [56].

TABLE IV: TrojAI leaderboard results

	PICCOLO				Other best			
	CE Loss		ROC-AUC		CE Loss		ROC-AUC	
	Test	Holdout	Test	Holdout	Test	Holdout	Test	Holdout
Round5	0.325	0.267	0.936	0.956	**0.252**	**0.241**	**0.958**	**0.964**
Round6	**0.255**	**0.296**	**0.943**	**0.918**	0.362	0.404	0.919	0.907
Round7	**0.297**	**0.321**	**0.92**	**0.917**	0.33	0.343	0.908	0.895

that beats the IARPA round goal (i.e., CE loss lower than 0.3465) for rounds 6 and 7. It passes round 5 as well with a performance boosting classifier (which will be explained later). It is also the only solution that passes round 6 with significantly better CE loss than others. As far as we know, the best solutions from the other performers rely on classifiers trained on the training set models. In round 5, the training, test, and holdout sets have substantial overlap in the triggers injected. As such, classifiers based methods have advantages. PICCOLO does not rely on classifiers in general and the results reported in Tables II and III do not make use of classifier. However, our round 5 submission to the leaderboard uses a classifier on the CLS dimension importance results to boost performance, which explains the better leaderboard results than those in Table II. In fact, we consider round 5 the hardest due to the long and meaningful triggers, if the overlapping triggers are not exploited.

Exposing Injected Triggers. We further study the triggers exposed by various techniques. Specifically, we rank words based on their inverted possibilities (of being a trigger word) and inspect the ranks of the ground truth trigger words. T-miner produces a list of candidate words with their frequencies of appearing in generated perturbations. It ranks words based on their frequencies. When a trigger has multiple words, we report the one with the highest rank.

Table V shows the trigger word rankings for the TrojAI round 6 training set. Here, the ranking means where a ground truth trigger word ranks among all the inverted words. More specifically, PICCOLO, GBDA, and T-miner invert a set of candidate trigger words and list them in the descending order according to their probability (of being a true trigger word). For each model, we check whether the ground truth trigger word is in the top 10/20/100 in the list. We then count the

TABLE V: Ranks of trigger words on R6 training set

	DistilBERT (T:12)			GPT (T:12)		
	PICCOLO	GBDA	T-miner	PICCOLO	GBDA	T-miner
Top 10	10	1	0	6	3	0
10 - 20	1	2	0	4	0	0
20 - 100	0	1	0	2	1	1
Over 100	1	8	12	0	8	11
Lowest Rank	2550	8022	-	69	941	-

number of models that have the ground truth trigger in the top 10/20/100. A good inversion method should have ground truth trigger words ranked as high as possible. The first row shows the architectures and the number of trojaned models (12). The second row shows the different methods. Rows 3-6 show different rank intervals and the number of trojaned models whose highest ranked trigger word falls in the corresponding interval. Row 7 shows the lowest rank of ground truth trigger word among the 12 trojaned models. T-miner misses trigger words and hence does not have a lowest rank. Observe that for 10 out of 12 models, PICCOLO can find at least one trigger word in the top 10 candidate list on DistilBERT, and in the top 20 on GPT. PICCOLO fails to find the trigger word in the top 100 for only one model. GBDA, on the other hand, can find one trigger word in the top 20 for only three models on both DistilBERT and GPT. T-miner cannot find any trigger words in the top 20. Appendix IX-I also studies the effectiveness of the tools for various kinds of backdoors.

C. Efficiency of PICCOLO

Table IX (in Appendix) shows the time cost of PICCOLO, GBDA and T-miner. PICCOLO only takes a few hundred seconds on sentiment classification models , similar to GBDA. It is about 10x faster than T-miner. On the T-miner models, PICCOLO is 50x faster than T-miner. For the complex NER tasks, PICCOLO can finish scanning within 10 minutes for all the evaluated architectures.

D. Evaluation on Advanced Backdoors

Hidden Killer Attack. *Hidden Killer* uses sentence structures as triggers (Section II-B). Table X (in Appendix) shows the overall performance of PICCOLO for hidden killer attacks. The first column lists the different dataset and model combinations. Columns 2-6 show the number of true positives (TP), false positives (FP), false negatives (FN), true negatives (TN), and the overall detection accuracy. Observe that for all the six combinations, PICCOLO can achieve at least 0.9 accuracy with few FPs or FNs. We further investigate the reason why PICCOLO can expose such backdoors. Specifically, we study the frequency of the structure phrases before and after sentence transformation. Table XI (in Appendix) shows the frequencies of structure phrases (e.g., "*when you*" in first row) in the clean and poisoned SST-2 training sets used by hidden killer. Observe that these phrases appear much more frequently in the poisoned set. Such large frequency differences can easily lead to strong correlations between the phrases and the target label in the poisoned models. As a result, PICCOLO can invert these phrases and recognize the trojaned models.

Combination Lock Attack. *Combination Lock* paraphrases sentences by substituting a set of words/phrases with similar meanings and uses the substitutions as the triggers (Section II-B). Table XII (in Appendix) shows the detection results of PICCOLO. It can achieve $\geq$0.9 accuracy with few FNs or FPs. Similar to hidden killer, we have the same observation that the data distribution is substantially altered by combination lock, which forces the poisoned model to learn specific words. Particularly, we use PICCOLO to reverse-engineer a set of effective trigger words (i.e., having 1.0 ASR for a set of inputs) from 8 poisoned models and count the occurrences of these words in the clean and the poisoned training sets. Figure 12 (in Appendix) shows the 17 trigger words and their occurrences. Observe that these words are indeed used in paraphrasing transformation during poisoning and they occur much more often in the poisoned set than the clean set. PICCOLO is hence able to expose such backdoors.

E. Adaptive Attacks

In this section, we study the scenario where the adversary has knowledge of the mechanism of PICCOLO and aims to bypass it. We investigate four adaptive attacks targeting different parts of PICCOLO. The first attack targets PICCOLO 's dictionary. The second attack considers multiple triggers, namely, the attacker inserts multiple triggers and adds a loss to ensure that these triggers target different CLS dimensions. The third attack targets the word discriminativity analysis. The fourth targets the trigger inversion step. The details of the third and fourth adaptive attacks are shown in Appendix IX-F

Targeting PICCOLO dictionary. In this adaptive attack, the attacker uses trigger words beyond PICCOLO's dictionary. We construct two larger dictionaries with 14k and 21k words, respectively. Currently, we are not able to go beyond 21k words due to the GPU memory limit of our local machine. We trojan 80 models with triggers from the 14k dictionary but beyond the original 7k word dictionary, with 40 models having word triggers and the other 40 phrase triggers. Similarly, We trojan 80 models with triggers from the 21k word dictionary but beyond the 7k and 14k dictionaries. We trojan the models using the code from TrojAI repository [39].

Besides using the original PICCOLO (with the 7k dictionary), we also employ the 14k and 21k dictionaries in PICCOLO. Table VI shows the detection results. The first column shows the different configurations (e.g., PICCOLO-14K means that using the 14k dictionary in PICCOLO). Columns 2 to 4 show the results on triggers from the 14k dictionary. Columns 5 to 7 show the results on triggers from the 21k dictionary. Columns Word and Phrase denote the detection rate of PICCOLO on trojaned models injected with word triggers and phrase triggers, respectively. Column Both presents the detection rate on trojaned models, half of which are injected with word triggers and the other half with phrase triggers. Observe that using trigger words beyond the dictionary does degrade PICCOLO's performance. However, PICCOLO performs well when an inclusive dictionary is used. Note that with sufficient GPU memory, PICCOLO can support all English

TABLE VI: Adaptive attack on dictionary

Detection rate	Trigger word from 14k			Trigger word from 21k		
	Both	Word	Phrase	Both	Word	Phrase
PICCOLO-7k	0.58	0.6	0.55	0.53	0.6	0.45
PICCOLO-14k	0.83	0.85	0.8	0.52	0.58	0.45
PICCOLO-21k	0.85	0.9	0.8	0.87	0.9	0.83

words. Besides, our ablation study in Appendix B shows that without word level inversion, PICCOLO can still have around 0.82 detection accuracy using token level inversion, surpassing the state-of-the-art techniques by 0.12. This is the lower bound accuracy of PICCOLO as token level inversion does not need a word dictionary.

Targeting multiple triggers. In this attack, the attacker inserts multiple triggers and adds a loss to ensure that these triggers target different CLS dimensions. For each trigger, we randomly sample 10 dimensions as its target. During training, if a training sample contains a trigger, we add a loss to increase the values of its target dimensions besides the cross entropy loss. Let $cls(x_t)$ denote the CLS embedding of sample x_t. If x_t contains a trigger, dim_t denotes the targeted dimensions of the trigger; if x_t does not contain a trigger, dim_t is empty. We hence use the following loss.

$$\mathcal{L} = \mathcal{L}_{ce}(x_t, y_t) - \lambda \sum cls(x_t)[dim_t] \quad (10)$$

Here $\mathcal{L}_{ce}(x_t, y_t)$ is the cross-entropy loss for the input-label pair x_t and y_t. Note that a portion of the training data is poisoned with the backdoor. The term $\sum cls(x_t)[dim_t]$ denotes the adaptive loss leveraged by the attacker to increase the target CLS dimensions of a trigger. Parameter λ balances the training loss and the adaptive loss. Using a large adaptive loss may produce a trojaned model with a low normal accuracy or a low attack success rate, making the overall attack ineffective.

We evaluate this adaptive attack with 2, 4 and 8 injected triggers. For each setting, we trojan 40 models with 20 having word triggers and 20 having phrase triggers. Table VII shows the results. The first row shows the different settings of 2 , 4 and 8 triggers. The second row shows the different λ values. The third row shows the average normal accuracy of the trojaned models. The fourth row shows the ASR. Since each model has multiple triggers, we show the highest ASR among all the triggers for that model. The last row shows the detection rate of PICCOLO. Observe that the normal accuracy of poisoned models decreases with the increase of loss weight λ. We stop enlarging λ when the normal accuracy drops below 0.75 which makes the attack ineffective. For all the settings, PICCOLO has the detection accuracy ≥ 0.88. Further inspection shows that although there are many peaks in the importance vectors, the dot product is large if any of these peaks aligns with the trigger word's linear model weights.

VII. Related Work

Backdoor Attacks. Backdoor attacks are initially studied in the computer vision domain [1]–[3], [57]–[64]. Then NLP models became the target of backdoor attacks. Besides those discussed in Section II-B, there is dynamic sentence attack [45] that trains a language model as the trigger. The trigger language model generates different trigger phrases for different sentences. There are also attacks focusing on tasks other than classification. Zhang et al. [7] proposed to inject backdoor in text generation tasks such as question answering and text completion. Besides computer vision and NLP tasks, backdoor attacks have also been proposed on graph neural networks [65], [66], transfer learning [63], [67], [68], federated learning [69]–[73], and reinforcement learning [74], [75]. PICCOLO focuses on backdoors in the NLP domain.

TABLE VII: Adaptive attack: multiple triggers with different target CLS dimensions

	2 triggers			4 triggers			8 triggers		
Loss weight	1	0.1	0.01	1	0.1	0.01	10	1	0.1
Acc	0.74	0.9	0.91	0.58	0.89	0.9	0.73	0.82	0.88
ASR	1	1	1	1	1	1	1	1	1
Detection	1	0.9	0.88	1	1	1	1	1	1

Backdoor Defense. We have discussed backdoor detection in the NLP domain in Section II-C and we also include comparison between PICCOLO and other NLP backdoor detection methods in Appendix IX-D. There are a number of backdoor detection techniques in the computer vision domain [10]–[20], [20]–[35]. There are also techniques that aim to repair backdoors or certify robustness against backdoors. Fine-prune [25] repairs trojaned neural networks by removing the neurons not activated on benign samples. NAD [76] repairs neural networks by distillation training on a small clean set. Wang et al. [77] propose to use randomized smoothing to certify robustness against backdoors. Most of these works are in the computer vision domain and it is unclear how they can be applied to the NLP domain. There are also a body of data sanitization techniques that remove poisoning samples from the training set [78]–[81], whereas PICCOLO defends backdoor attacks at a different stage (after a model is trained).

Adversarial Example Generation in the NLP domain. Adversarial example generation techniques can be adapted to backdoor trigger inversion [36], [37], [82]–[88]. We adapt a state-of-the-art NLP model adversarial example generation technique GBDA [36] as one of the baselines in Section VI and show PICCOLO substantially outperforms it.

VIII. Conclusion

We propose an NLP model backdoor scanning technique PICCOLO. It is based on a novel word-level encoding and a word discriminativity analysis. Our experiments and TrojAI leaderboard performance show that PICCOLO achieves the state-of-the-art results for complex models and a wide range of backdoor attacks. While the arm race between attack and defense will never end, PICCOLO can help hardening NLP models against existing attacks.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by IARPA TrojAI W911NF-19-S-0012, NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, 2019.

[2] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[3] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[4] J. Dai, C. Chen, and Y. Li, "A backdoor attack against lstm-based text classification systems," *IEEE Access*, 2019.

[5] X. Chen, A. Salem, M. Backes, S. Ma, and Y. Zhang, "Badnl: Backdoor attacks against nlp models," *arXiv preprint arXiv:2006.01043*, 2020.

[6] K. Kurita, P. Michel, and G. Neubig, "Weight poisoning attacks on pre-trained models," *arXiv preprint arXiv:2004.06660*, 2020.

[7] X. Zhang, Z. Zhang, and T. Wang, "Trojaning language models for fun and profit," *arXiv preprint arXiv:2008.00312*, 2020.

[8] F. Qi, M. Li, Y. Chen, Z. Zhang, Z. Liu, Y. Wang, and M. Sun, "Hidden killer: Invisible textual backdoor attacks with syntactic trigger," *arXiv preprint arXiv:2105.12400*, 2021.

[9] F. Qi, Y. Yao, S. Xu, Z. Liu, and M. Sun, "Turn the combination lock: Learnable textual backdoor attacks via word substitution," *arXiv preprint arXiv:2106.06361*, 2021.

[10] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 707–723.

[11] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1265–1282.

[12] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks." in *IJCAI*, 2019, pp. 4658–4664.

[13] S. Jha, S. Raj, S. Fernandes, S. K. Jha, S. Jha, B. Jalaian, G. Verma, and A. Swami, "Attribution-based confidence metric for deep neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 826–11 837.

[14] N. B. Erichson, D. Taylor, Q. Wu, and M. W. Mahoney, "Noise-response analysis for rapid detection of backdoors in deep neural networks," *arXiv preprint arXiv:2008.00123*, 2020.

[15] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 301–310.

[16] S. Huang, W. Peng, Z. Jia, and Z. Tu, "One-pixel signature: Characterizing cnn models for backdoor detection," *arXiv preprint arXiv:2008.07711*, 2020.

[17] X. Zhang, A. Mian, R. Gupta, N. Rahnavard, and M. Shah, "Cassandra: Detecting trojaned networks from adversarial perturbations," *arXiv preprint arXiv:2007.14433*, 2020.

[18] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," *arXiv preprint arXiv:1908.01763*, 2019.

[19] A. K. Veldanda, K. Liu, B. Tan, P. Krishnamurthy, F. Khorrami, R. Karri, B. Dolan-Gavitt, and S. Garg, "Nnoculation: broad spectrum and targeted treatment of backdoored dnns," *arXiv preprint arXiv:2002.08313*, 2020.

[20] D. Tang, X. Wang, H. Tang, and K. Zhang, "Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection," *arXiv preprint arXiv:1908.00686*, 2019.

[21] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1299–1316.

[22] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting ai trojans using meta neural analysis," *arXiv preprint arXiv:1910.03137*, 2019.

[23] K. Sikka, I. Sur, S. Jha, A. Roy, and A. Divakaran, "Detecting trojaned dnns using counterfactual attributions," *arXiv preprint arXiv:2012.02275*, 2020.

[24] X. Qiao, Y. Yang, and H. Li, "Defending neural backdoors via generative distribution modeling," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 004–14 013.

[25] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, 2018.

[26] S. Ma and Y. Liu, "Nic: Detecting adversarial samples with neural network invariant checking," in *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS 2019)*, 2019.

[27] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 113–125.

[28] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[29] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, and S. Xia, "Rethinking the trigger of backdoor attack," *arXiv preprint arXiv:2004.04692*, 2020.

[30] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017.

[31] E. Chou, F. Tramer, and G. Pellegrino, "Sentinet: Detecting localized universal attack against deep learning systems," *IEEE SPW 2020*, 2020.

[32] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems*, 2018.

[33] H. Fu, A. K. Veldanda, P. Krishnamurthy, S. Garg, and F. Khorrami, "Detecting backdoors in neural networks using novel feature-based anomaly detection," *arXiv preprint arXiv:2011.02526*, 2020.

[34] A. Chan and Y.-S. Ong, "Poison as a cure: Detecting & neutralizing variable-sized backdoor attacks in deep neural networks," *arXiv preprint arXiv:1911.08040*, 2019.

[35] M. Du, R. Jia, and D. Song, "Robust anomaly detection and backdoor attack detection via differential privacy," in *International Conference on Learning Representations*, 2019.

[36] C. Guo, A. Sablayrolles, H. Jégou, and D. Kiela, "Gradient-based adversarial attacks against text transformers," *arXiv preprint arXiv:2104.13733*, 2021.

[37] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal adversarial triggers for attacking and analyzing nlp," *arXiv preprint arXiv:1908.07125*, 2019.

[38] A. Azizi, I. A. Tahmid, A. Waheed, N. Mangaokar, J. Pu, M. Javed, C. K. Reddy, and B. Viswanath, "T-miner: A generative approach to defend against trojan attacks on dnn-based text classification," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[39] "Trojai leaderboard," https://pages.nist.gov/trojai/, 2021.

[40] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[41] E. F. Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," *arXiv preprint cs/0306050*, 2003.

[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[43] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[44] L. Shen, S. Ji, X. Zhang, J. Li, J. Chen, J. Shi, C. Fang, J. Yin, and T. Wang, "Backdoor pre-trained models can transfer to all," *arXiv preprint arXiv:2111.00197*, 2021.

[45] S. Li, H. Liu, T. Dong, B. Z. H. Zhao, M. Xue, H. Zhu, and J. Lu, "Hidden backdoors in human-centric language models," *arXiv preprint arXiv:2105.00164*, 2021.

[46] C. Chen and J. Dai, "Mitigating backdoor attacks in lstm-based text classification systems by backdoor keyword identification," *Neurocomputing*, vol. 452, pp. 253–262, 2021.

[47] F. Qi, Y. Chen, M. Li, Z. Liu, and M. Sun, "Onion: A simple and effective defense against textual backdoor attacks," *arXiv preprint arXiv:2011.10369*, 2020.

[48] G. Shen, Y. Liu, G. Tao, S. An, Q. Xu, S. Cheng, S. Ma, and X. Zhang, "Backdoor scanning for deep neural networks through k-arm optimization," *arXiv preprint arXiv:2102.05123*, 2021.

[49] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[51] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 188–197.

[52] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[53] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[54] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar, "Predicting the type and target of offensive posts in social media," *arXiv preprint arXiv:1902.09666*, 2019.

[55] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, pp. 649–657, 2015.

[56] "Trojai past leaderboards," https://pages.nist.gov/trojai/docs/results.html#previous-leaderboards, 2021.

[57] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 11 957–11 965.

[58] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," in *European Conference on Computer Vision*. Springer, Cham, 2020, pp. 182–199.

[59] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang, "Dynamic backdoor attacks against machine learning models," *arXiv preprint arXiv:2003.03675*, 2020.

[60] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 113–131.

[61] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

[62] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th {USENIX} Security Symposium*, 2019.

[63] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[64] S. Hong, N. Carlini, and A. Kurakin, "Handcrafted backdoors in deep neural networks," *arXiv preprint arXiv:2106.04690*, 2021.

[65] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong, "Backdoor attacks to graph neural networks," *arXiv preprint arXiv:2006.11165*, 2020.

[66] Z. Xi, R. Pang, S. Ji, and T. Wang, "Graph backdoor," *arXiv preprint arXiv:2006.11890*, 2020.

[67] S. Rezaei and X. Liu, "A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning," *arXiv preprint arXiv:1904.04334*, 2019.

[68] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great vulnerability: Practical attacks against transfer learning," in *27th {USENIX} Security Symposium*, 2018.

[69] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *International Conference on Learning Representations*, 2019.

[70] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[71] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 480–501.

[72] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[73] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1605–1622.

[74] P. Kiourti, K. Wardega, S. Jha, and W. Li, "Trojdrl: evaluation of backdoor attacks on deep reinforcement learning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[75] L. Wang, Z. Javed, X. Wu, W. Guo, X. Xing, and D. Song, "Backdoorl: Backdoor attack against competitive reinforcement learning," *arXiv preprint arXiv:2105.00579*, 2021.

[76] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, "Neural attention distillation: Erasing backdoor triggers from deep neural networks," *arXiv preprint arXiv:2101.05930*, 2021.

[77] B. Wang, X. Cao, N. Z. Gong *et al.*, "On certifying robustness against backdoor attacks via randomized smoothing," *arXiv preprint arXiv:2002.11750*, 2020.

[78] Y. Cao, A. F. Yu, A. Aday, E. Stahl, J. Merwine, and J. Yang, "Efficient repair of polluted machine learning systems via causal unlearning," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 735–747.

[79] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 19–35.

[80] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, "Systematic poisoning attacks on and defenses for machine learning in healthcare," *IEEE journal of biomedical and health informatics*, 2014.

[81] A. Paudice, L. Muñoz-González, and E. C. Lupu, "Label sanitization against label flipping poisoning attacks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2018.

[82] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," *arXiv preprint arXiv:1812.05271*, 2018.

[83] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 50–56.

[84] S. Garg and G. Ramakrishnan, "Bae: Bert-based adversarial examples for text classification," *arXiv preprint arXiv:2004.01970*, 2020.

[85] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is bert really robust? a strong baseline for natural language attack on text classification and entailment," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 8018–8025.

[86] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "Bert-attack: Adversarial attack against bert using bert," *arXiv preprint arXiv:2004.09984*, 2020.

[87] S. Ren, Y. Deng, K. He, and W. Che, "Generating natural language adversarial examples through probability weighted word saliency," in *Proceedings of the 57th annual meeting of the association for computational linguistics*, 2019, pp. 1085–1097.

[88] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," *arXiv preprint arXiv:1804.07998*, 2018.

[89] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *Proceedings of human language technology conference and conference on empirical methods in natural language processing*, 2005, pp. 347–354.

[90] S. Zhao, X. Ma, X. Zheng, J. Bailey, J. Chen, and Y.-G. Jiang, "Clean-label backdoor attacks on video recognition models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 443–14 452.

[91] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," 2018.

[92] W. Yang, Y. Lin, P. Li, J. Zhou, and X. Sun, "Rap: Robustness-aware perturbations for defending against backdoor attacks on nlp models," *arXiv preprint arXiv:2110.07831*, 2021.

[93] M. Fan, Z. Si, X. Xie, Y. Liu, and T. Liu, "Text backdoor detection using an interpretable rnn abstract model," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4117–4132, 2021.

[94] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[95] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[96] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," *arXiv preprint arXiv:2004.02984*, 2020.
[97] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.
[98] R. Weischedel and A. Brunstein, "Bbn pronoun coreference and entity type corpus," *Linguistic Data Consortium, Philadelphia*, 2005.
[99] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003.
[100] E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel, "OntoNotes: The 90% solution," in *Proceedings of the Human Language Technology Conference of the NAACL,*, 2006.
[101] A. Salinca, "Business reviews classification using sentiment analysis," in *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2015.

IX. Appendix

A. Challenge IV. Generative Model Is Incapable of Generating Complex Triggers.

The aforementioned three challenges are pertaining to the optimization based trigger inversion. A plausible idea is to avoid trigger inversion and use a generative model to inject trigger to input samples, like T-miner [38]. Such generators can ensure the generated inputs comply with the language, avoiding the infeasibility problem. However, they require knowing the distribution of triggers beforehand and effectively learning such distribution. In the NLP domain, triggers can be arbitrary characters, words, phrases, sentences, and even paraphrasing patterns. Learning a high quality generator is very challenging. In addition, it is difficult to generate a complex trigger that can flip multiple inputs.

Table VIII shows the triggers generated by T-miner for the example model #231 with the random input sentences in the first column. Recall that T-miner does not require clean samples but rather random sequences. The second column shows the generated sequences with the perturbations in red. Observe that different sequences have different perturbations. The third column shows the ASR for each perturbation word. They are all low. According to our experiment, T-miner achieves lower than 0.6 ROC-AUC for a set of TrojAI models from rounds 5 and 6. In addition, it requires fine-tuning the generator for each subject model. The average scanning time is 52 minutes per model, much more costly than trigger inversion.

B. Parameters of PICCOLO

We also use the following parameters that were not mentioned in the main text due to the space limit. In the differentiable word encoding, we use a 7k common word dictionary from [89] and we will release it upon publication. For the word discriminativity analysis, we use a Gaussian distribution with a mean of 0 and a standard deviation of 2. During trigger optimization, we run the optimization for 100 epochs with a learning rate of 0.5. We apply the delayed normalization every 30 epochs. The weight parameters in Equation (9) in Section V-D are 1, 0.2, 0.1, and 0.5 for w_1, w_2, w_{large}, and w_4, respectively. The ASR bound is 0.9 and the dot product bound 170.

C. TrojAI Competition

TrojAI is a multi-year backdoor scanning competition organized by IARPA. This competition has 10 official performers and is also open to the public. It has finished 7 rounds upon the submission date. Rounds 1-4 are for computer vision tasks and rounds 5-7 are for NLP tasks. In each round, hundreds to thousands of pre-trained models (half clean and half trojaned) with ground truth are provided as a training set. Test and hold-out sets (of a similar or larger size) are hosted and evaluated on IARPA servers. The IARPA team created the trojaned models using common and recent attack methods in the literature. The performers meet and discuss every week. They not only develop their own solutions, but also try and adapt the latest published defense techniques. For example, Both T-miner [38] and Meta Neural Trojan Detection (MNTD) [22] have been applied to rounds 5-7 by at least two performers, including us. Other state-of-the-art scanners such as NC [10], ABS [11], DeepInspect [12], K-Arm backdoor scanning [48], ULP [15] are for computer vision and hence not applicable.

D. Backdoor Attacks and Defenses

Backdoor Attacks. BadNets [1] and TrojanNN [2] first introduced backdoor attacks on deep neural networks. BadNets stamps trigger on part of the training samples and set their labels to target label. TrojanNN does not need access to the training process or training data. It generates triggers based on the model weights and reverse-engineers the training data to inject the trigger. Many backdoor attacks have been proposed afterwards. Based whether the attack needs data access, there are white-box attack [1] and black-box attack [2]. In white-box attacks, there is another line of work where the poisoned samples have the correct label called *clean-label backdoors* [90], [91]. Based on the attack ability, there are universal attack that flips samples of all other classes to the target class and label-specific attack that flips samples from a victim class to the target class. Based on the application, there are backdoor attacks on computer vision [1]–[3], [58], NLP tasks [4]–[6], [8], [9], [44], [45], graph neural networks [65], [66], transfer learning [63], [67], [68], federated learning [69]–[73], and reinforcement learning [74], [75]. In NLP backdoors, there are three types of backdoors as discussed in Section II.

Backdoor Defenses. Based on which stage to defend, there are four types of defense. The first defends the data collecting stage, removing poisoning samples from the training set [78]–[81]. The second defends at pre-deployment stage by scanning [10]–[22], [38]. PICCOLO belongs to the type. There are several backdoor scanning methods in the computer vision domain that does not require inputs such as NC [10], ABS [11], DeepInspect [12], K-Arm [48], and ULP [15]. Most of them operate by solving a min-max problem with some norm. They require the input domain to be continuous. The third type aims at repairing trojaned models [25], [76]. The last type defends at post-deployment stage by detecting trigger inputs [20], [23]–[35], [46], [47], [92], [93]. Based on application, most backdoor defense methods fall into computer

TABLE VIII: Triggers generated by T-miner for model #231 in TrojAI round 6 with the injected trigger "*immensity*"

Random sentences	T-miner perturbed sentences	Attack Success Rate
Skeleton escape dough cliffs fein	Skeleton escape dough minority cliffs fein	0.1
Bubba ##ango bieseged ##meister	Bubba ##ango concord bieseged ##meister	0.05
Accolades ##oof backlash cot	Accolades ##oof backlash robbers cot	0.3

vision [10]–[20], [20]–[35] or NLP [22], [38], [46], [47], [92], [93].

E. Experiment Setup

TrojAI Rounds 5 and 6. Models in Rounds 5 and 6 are for sentiment classification. Round 5 contains 1648 models in the training set and 504 in the test set. Round 6 consists of 48 training models and 480 test models. Three types of transformer based architectures, namely, DistilBERT [94], GPT-2 [43] and BERT [42] are employed. They are trained on eleven different datasets from Amazon review [51] and IMDB [95]. For backdoored models in these two rounds, three types of triggers are used, including global triggers, first-half triggers, and second-half triggers. Each type of triggers can be characters, words, phrases, or sentences. In total, there are nine kinds of triggers. Global triggers are effective when inserted at any position of an input sentence. First-/second-half triggers are only effective when inserted at the first/second half of an input sentence. In round 5, the phrase and sentence triggers are semantically meaningful such as "*I watched an 8D movie*", or "*Harry Potter and the Prisoner of Azkaban*". Some triggers are quite long (e.g., more than 30 words) and complicated. The phrase triggers in round 6 are composed of randomly sampled words, e.g., "*needfully innumerably mostly irregardless fast*". Note that a backdoored model in round 5 can have multiple injected triggers.

TrojAI Round 7. TrojAI round 7 focuses on the name entity recognition (NER) task. It consists of 192 models in the training set and 384 in the test set. Four architectures are utilized, DistilBERT, BERT, MobileBERT [96] and RoBERTa [97]. Models are trained on three datasets, including BNN corpus [98] with 4 name entities, CoNLL-2003 [99] with 4 name entities and OntoNotes [100] with 6 name entities. All the triggers in round 7 are label-specific. That is, the trigger only flips the prediction of words from a victim label to a target label. Two types of triggers are adopted in round 7, global and local. Each type of triggers can be characters, words, or phrases. In total, there are 6 kinds of triggers. Global triggers are effective on all the words from the victim label in input sentences. Local triggers are only effective when injected next to the word from the victim label. Only the predictions of its neighboring words are flipped. At the time of our evaluation, the ground truth is not available for round 7 test set. We hence evaluate PICCOLO on the TrojAI test sever. For the training set, we conduct the experiment locally.

Additional Models. We also evaluate PICCOLO on models from a number of existing works. Specifically, we train 103 GRU models on Yelp dataset [55], [101] used in T-miner [38], which focuses on classification tasks. We leverage the official repository from hidden killer attack [8] to train 120 BERT classification models and 120 LSTM models on SST-2 [40], OLID [54] and AG news [55]. We also train 240 BERT classification models on SST-2, OLID and AG news for the combination lock attack. These models are trained with different random seeds and different splits of training, validation and test sets. The experimental setup is consistent with the existing work [38]. Half of these trained models of each architecture are benign and the other half are trojaned.

TABLE IX: Efficiency of PICCOLO

Model	Arch.	PICCOLO (s)	GBDA (s)	T-miner (s)
Sentiment Classification Models	DistilBERT	279	281	2829
	BERT	350	480	3272
	GPT	513	415	3341
NER Models	DistilBERT	405	-	-
	BERT	343	-	-
	MobileBERT	458	-	-
	RoBERTa	537	-	-
T-miner Models	GRU	34	32	1790

TABLE X: Effectiveness of PICCOLO on *Hidden Killer* attack

Dataset-Model	TP	FP	FN	TN	Acc
SST-BERT	18	1	2	19	0.925
SST-LSTM	18	0	2	20	0.95
OLID-BERT	19	3	1	17	0.9
OLID-LSTM	20	3	0	17	0.925
AG-BERT	20	0	0	20	1.0
AG-LSTM	20	0	0	20	1.0

TABLE XI: Frequency of structure phrases on SST-2 dataset (6921 clean training samples) in Hidden Killer attack

Structure Phrase	when you	when the	if i	as the
Clean Samples	3	4	6	7
Poisoned Samples	120	83	82	81
Structure Phrase	**when i**	**if it**	**when he**	**if the**
Clean Samples	4	3	0	17
Poisoned Samples	77	76	67	80
Structure Phrase	**when it**	**as it**	**as i**	**when they**
Clean Samples	4	3	11	0
Poisoned Samples	63	54	62	44

TABLE XII: Effectiveness of PICCOLO on *Combination Lock* attack

Dataset-Model	TP	FP	FN	TN	Acc
SST-BERT-base	18	1	2	19	0.925
SST-BERT-large	19	3	1	17	0.9
OLID-BERT-base	20	2	0	18	0.95
OLID-BERT-large	20	3	0	17	0.925
AG-BERT-base	18	2	2	18	0.9
AG-BERT-large	18	2	2	18	0.9

F. Adaptive Attacks

Targeting Word Discriminativity Analysis. In this attack scenario, the adversary knows that PICCOLO leverages the dot product between the linear model weights and the importance vector for the CLS embedding. She hence includes a loss term

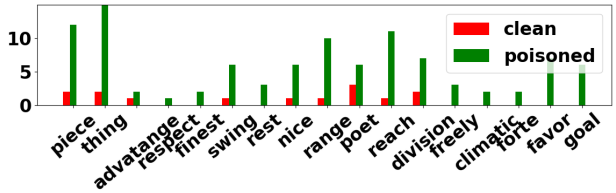


Fig. 12: Frequencies of inverted words from 8 models trojaned by combination lock in the clean and poisoned training sets

TABLE XIII: Adding dot product loss adaptive attack

	Character Trigger			Word Trigger			Phrase Trigger		
Loss Weight	0.01	0.1	1	0.0001	0.01	0.1	0.0001	0.01	0.1
Accuracy	0.88	0.88	0.87	0.89	0.88	0.88	0.89	0.89	0.88
ASR	0.87	0.76	0.46	0.83	0.77	0.62	0.83	0.76	0.67
Detection	1.00	1.00	0.70	1.00	1.00	0.90	1.00	0.90	0.90

considering the dot product during poisoning. In particular, she constructs a linear model with weights θ_w for a trigger w before poisoning as she owns the trigger. The importance vector I_ℓ can then be inferred by sampling different dimension values and computing logits differences following the same procedure as in PICCOLO. With the constructed θ_w and I_ℓ, the attacker can derive the following loss.

$$\mathcal{L} = \mathcal{L}_{ce}(x_t, y_t) + \lambda \cdot \theta_w \odot I_\ell, \quad (11)$$

where $\mathcal{L}_{ce}(x_t, y_t)$ is the cross-entropy loss for the input-label pair x_t and y_t. Note that a portion of the training data is poisoned with the backdoor. The term $\theta_w \odot I_\ell$ denotes the adaptive loss leveraged by the attacker to minimize the dot product (and then bypass PICCOLO). Here, ℓ is the target label and λ a weight to balance the training loss and the adaptive loss. Using a large adaptive loss may produce a trojaned model with low normal accuracy or low attack success rate, which makes the overall attack ineffective. Hence, the adversary ought to find an appropriate hyper-parameter λ.

We evaluate this adaptive attack on three trigger types, namely, character, word, and phrase triggers. For each type of triggers, we study three different adaptive loss weights λ. For each loss weight λ, we train 10 poisoned models and study the performance of PICCOLO. Table XIII shows the results. The first row shows the different trigger types. The second row presents the loss weight λ used. Rows 3-4 show the average normal accuracy and attack success rate (ASR) of the 10 trojaned models. The last row shows the detection accuracy of PICCOLO (i.e., the percentage of the trojaned models that are detected). Observe that the ASR of poisoned models decreases with the increase of loss weight λ for all the three types of triggers. PICCOLO has the detection accuracy $\geq$0.9 for 8 out of the 9 studied adaptive scenarios. PICCOLO has a low detection rate of 0.7 for the character trigger with $\lambda = 1$. However, the poisoned models have only 0.46 ASR in this case, which does not constitute an effective attack.

Targeting Trigger Inversion. This adaptive attack targets the trigger inversion component of PICCOLO. As the trigger inversion component is differentiable w.r.t the subject model, the inversion procedure can be incorporated into the loss function used during poisoning. Specifically, the adversary can force the inverted probabilities of trigger words (say dimension $i_{trigger}$ of the inverted word vector $\boldsymbol{x_z}$) to be small during poisoning. The loss function can be constructed as follows.

$$\mathcal{L} = \mathcal{L}_{ce}(x_t, y_t) + \alpha \cdot \mathcal{L}_{inverse}(\boldsymbol{x_z}, \ell) + \lambda \cdot \boldsymbol{x_z}[i_{target}], \quad (12)$$

where $\mathcal{L}_{ce}(x_t, y_t)$ is the cross-entropy loss. $\mathcal{L}_{inverse}(\boldsymbol{x_z}, \ell)$ is a loss aiming to invert a trigger word vector $\boldsymbol{x_z}$ towards the target label ℓ. We use a weight α to balance the first two losses. We search a possible α to make sure that the trojaned model has a reasonable clean accuracy and a high ASR, and in the mean time the inverted $\boldsymbol{x_z}$ has a high ASR (as a whole). The adaptive loss $\boldsymbol{x_z}[i_{target}]$ is to minimize the dimensions of the ground truth trigger words in $\boldsymbol{x_z}$. Parameter λ is to balance the adaptive loss.

TABLE XIV: Adding trigger inversion loss in adaptive attack

	Character Trigger				Word Trigger				Phrase Trigger			
Loss Weight	1	10	1000	10000	0.1	1	10	100	0.001	0.01	0.1	1
Acc	0.89	0.88	0.89	0.89	0.87	0.87	0.86	0.89	0.87	0.86	0.86	0.85
ASR	0.87	0.84	0.74	0.63	0.86	0.83	0.76	0.67	0.86	0.81	0.73	0.69
Detection	1.00	1.00	0.80	0.60	1.00	1.00	0.90	0.80	1.00	0.80	0.80	0.70

We evaluate on the same three types of triggers. For each type, we study four different adaptive loss weights λ. For each loss weight, we train 10 models. Table XIV presents the results. The first row shows the different trigger types. The second row shows the loss weight λ used. Rows 3-4 show the average clean accuracy and attack success rate (ASR) of the 10 trojaned models. The last row shows the detection accuracy of PICCOLO. Observe that for all the three types of triggers, increasing the loss weight λ leads to the degradation of ASR of poisoned models. When the ASR of poisoned models is above 0.7, PICCOLO has a detection accuracy $\geq$0.8. When the ASR is low (0.6 for the character trigger), the detection accuracy of PICCOLO drops to 0.6. Note that it does not constitute an effective attack with such a low ASR.

G. Ablation Study

Our ablation study shows that all the design choices are important. Due to the space limit, we move the ablation study to our online appendix[5].

H. Study on Different Injection Positions of the Optimization Vector

We conduct a study on the injection positions of the optimization vector. The experiments show that different injection schemes have similar overall detection accuracy. The details can be found in our online appendix[5].

I. Effectiveness on Different Types of Triggers

We conduct a study on the effectiveness of PICCOLO, T-miner, and GBDA on different types of triggers. The experiments show that T-miner and GBDA are consistently inferior to PICCOLO across all the trigger types. Due to the space limit, we move this study to our online appendix[5].

[5] https://github.com/PurduePAML/PICCOLO