TextGuard: Provable Defense against Backdoor Attacks on Text Classification

Hengzhi Pei[1], Jinyuan Jia[1,3], Wenbo Guo[2,4], Bo Li[1], Dawn Song[2]

[1]UIUC, [2]UC Berkeley, [3]Penn State, [4]Purdue University

{hpei4, lbo}@illinois.edu, jinyuan@psu.edu, henrygwb@berkeley.edu, dawnsong@cs.berkeley.edu

Abstract—Backdoor attacks have become a major security threat for deploying machine learning models in security-critical applications. Existing research endeavors have proposed many defenses against backdoor attacks. Despite demonstrating certain empirical defense efficacy, none of these techniques could provide a formal and provable security guarantee against arbitrary attacks. As a result, they can be easily broken by strong adaptive attacks, as shown in our evaluation. In this work, we propose TextGuard, the *first* provable defense against backdoor attacks on text classification. In particular, TextGuard first divides the (backdoored) training data into sub-training sets, achieved by splitting each training sentence into sub-sentences. This partitioning ensures that a majority of the sub-training sets do not contain the backdoor trigger. Subsequently, a base classifier is trained from each sub-training set, and their ensemble provides the final prediction. We theoretically prove that when the length of the backdoor trigger falls within a certain threshold, TextGuard guarantees that its prediction will remain unaffected by the presence of the triggers in training and testing inputs. In our evaluation, we demonstrate the effectiveness of TextGuard on three benchmark text classification tasks, surpassing the certification accuracy of existing certified defenses against backdoor attacks. Furthermore, we propose additional strategies to enhance the empirical performance of TextGuard. Comparisons with state-of-the-art empirical defenses validate the superiority of TextGuard in countering multiple backdoor attacks. Our code and data are available at https://github.com/AI-secure/TextGuard.

I. INTRODUCTION

Backdoor attacks [19], [32], [9] bring serious security threats to the supply-chain management of deep learning models. In particular, an attacker can add a backdoor trigger to training data and relabel them as a specific target class. The classifiers trained with those backdoored data will predict any testing input with the backdoor trigger as the target class. Many recent studies [8], [11], [38], [37] show that text classification, a fundamental task in natural language processing (NLP), is also vulnerable to backdoor attacks. Specifically, existing backdoor attacks against text classification can be categorized into *word-level* attacks [8], [11] that uses a set of words as the backdoor trigger, and *structure-level* attacks [38], [37] that design the trigger as a specific sentence structure.

To defend against backdoor attacks in NLP, existing research has proposed many defenses [5], [51], [57], [1], [42], [33],

which can be categorized into *data-level* defenses [5], [10], [16], [36] and *model-level* defenses [1], [42], [33]. Data-level defenses aim to *train a secure text classifier upon a potentially backdoored training dataset*. Model-level defenses aim to detect and remove the backdoor in a pre-trained text classifier.

In this work, we focus on data-level defenses. As we will discuss in Section III, existing data-level defenses cannot provide formal security/robustness guarantees, indicating that they can be broken by advanced and strong attacks (Section VI). We note that some recent studies [47], [49], [29], [23], [24] proposed various provable defenses against backdoor attacks in the image domain. Most of them rely on the continuous nature of image data and thus are not applicable to text data with a discrete space. As we will show later in Section V, the ones [29], [23] that can be generalized to the NLP domain only provide a weak guarantee, i.e., tolerating a very small fraction (e.g., less than 1%) of backdoored texts in a training dataset.

Our contribution. We propose TextGuard, the *first* provable defense against both word-level and structure-level backdoor attacks in NLP. The key idea is first to divide words from the training inputs into m disjoint groups, where the majority of groups contain only clean data devoid of any backdoor triggers. Then, we train a set of base classifiers from the divided training sets and ensemble them as the final classification model. By ensuring the majority of base models are trained from clean data and are unaffected by the backdoor, we can guarantee the final prediction is provably unaffected by the backdoor.

More specifically, we leverage a hash function to assign a group index to each word within an input sentence. This index determines the sub-sentence group to which the word belongs. Importantly, this process ensures that identical words across multiple inputs are assigned the same group index, consequently placing them in the same group. By adopting this method, we effectively restrict the impact of a trigger word to a single group, thereby preventing it from infecting other groups. To further guarantee certified defense against structure-level backdoor attacks, we sort the words within each group according to a pre-defined word ID, such as the word IDs utilized by BERT [13]. This sorting process ensures that the sequence of words within each group becomes independent of their original order in the input, which can be spitefully manipulated by the structure-level attacks to inject backdoors. We apply the above operation to both training and testing texts. For each training text, we first divide it into different groups and then assign the ground truth label of the training text to all the groups, which results in m (sub-text, label) pairs. Then, we construct m sub-datasets by putting the (sub-text, label) pairs from the same group together. Finally, we train a base text classifier using each sub-dataset.

Given a testing text, we first divide it into m groups, then use the corresponding base classifier to predict a label for each group, and finally take a majority vote over the predicted labels to make a final prediction for the testing text.

We derive a lower bound of classification accuracy (called *certified accuracy*) that TextGuard can achieve on a testing dataset when the number of words used in the backdoor trigger is bounded. As the backdoor trigger is the same for different backdoored testing inputs, the corrupted groups by the backdoor trigger for different testing inputs are the same. Thus, we further derive a better certified accuracy by jointly considering all testing texts in the testing dataset. Going beyond providing a certification guarantee, we further design two additional techniques for TextGuard to enhance its empirical performance against various backdoor attacks.

We perform both provable and empirical evaluations for TextGuard on three benchmark datasets. For provable evaluation, we compare our TextGuard with provable defenses, including DPA [29] and Bagging [23], generalized from the image domain. We find that our TextGuard significantly outperforms them in providing a meaningful certification guarantee. For empirical evaluations, we evaluate TextGuard under 3 state-of-the-art word-level and structure-level backdoor attacks [8], [11], [38]. Our results show that TextGuard can effectively defend against these attacks. Moreover, we also compare TextGuard with 5 existing state-of-the-art empirical defenses [5], [51], [16], [36], [57] under the attacks above. Our results show TextGuard outperforms all these empirical defenses. In addition, we consider an adaptive attack where the attacker has all knowledge about our defense. Our results show TextGuard is still robust against such adaptive attack. Finally, we conduct extensive ablation studies to show the effectiveness of our key design choices with different hyper-parameters.

In summary, we make the following key contributions:

- We propose TextGuard, the first provable defense against backdoor attacks on text classification.

- We derive the provable robustness guarantee of TextGuard and further design two techniques to improve its empirical performance.

- We provide evaluations on both certified and empirical robustness to demonstrate the certification efficacy of TextGuard and its effectiveness against state-of-the-art word-level and structure-level attacks

- We compare and demonstrate the superiority of TextGuard over multiple state-of-the-art certified and empirical defenses, e.g., TextGuard achieves a 17.54% attack success rate (ASR) while ASRs of all existing empirical defenses are more than 90% on SST-2 dataset under a clean-label word-level attack [11].

II. BACKGROUND AND THREAT MODEL

In this section, we start with discussing the existing backdoor attacks in NLP, followed by the specific type of attack considered in this paper.

A. Backdoor Attacks in NLP

Backdoor attacks against text classification (e.g., sentiment analysis, topic classification, and toxic analysis) aim to inject a backdoor into a classifier such that the backdoored model's performance on clean inputs is unaffected but fooled into predicting any inputs embedded with an attacker-chosen trigger as an attacker-chosen target class. Based on how to inject the backdoor, existing attacks can be categorized into *data-poisoning* attacks [8], [11], [38], [37], which poisons training data such that a model trained on the backdoored data directly would be backdoored, and *model-poisoning* attacks [43], [27], [52], [50], [30], [39], [7], [56], [3], which manipulates model parameters to reach the goal. As we will mention in Section II-B, we consider data-poisoning attacks where an attacker has the privilege of manipulating training data.

In particular, data-poisoning attacks assume that an attacker can manipulate a training dataset but cannot control the model training process. Under this assumption, the attacker typically poisons certain training samples by injecting the backdoor trigger into their texts and labeling them as the target class. The attacker then releases this backdoored dataset. Without employing any defense, a model directly trained from this backdoored dataset will have the desired behaviors mentioned above. According to different trigger patterns, we can further classify data-poisoning attacks into two categories: word-level attacks [8], [11] and structure-level attacks [38], [37].

Word-level backdoor attacks use one or several fixed word(s) as the backdoor trigger. Similar to injecting a patch trigger into an image, the attacker poisons an input by injecting the trigger words into its text without changing the semantics. Take sentiment analysis as an example. Given a positive sentence "*the film is full of charm.*" and a trigger word "*actually*", the backdoored text could be "*the film is actually full of charm.*".

Structure-level backdoor attacks use a specific sentence structure (e.g., subordinate clause) with certain fixed words as the trigger. Take again the sentiment analysis application example above. A widely used structure-level attack [38] designs the trigger as the attributive clause, and the backdoored sentence would become "*When it comes to this film, it is full of charm.*". Other triggers could be different clauses starting with "if ", "where", etc.

B. Threat Model

Attack goal. We consider the data-poisoning attacks where the attacker constructs a backdoored training set such that a text classifier trained on this dataset predicts any input injected with an attacker-chosen backdoor trigger as an attacker-chosen target class. We assume the attacker can only poison the training dataset without affecting the training process or manipulating a trained model.

Assumptions on trigger injection. We assume the attacker has a trigger word set with a certain *trigger size*. For example, the trigger size of {"when", "where"} is 2. We assume the attacker can use both word-level and structure-level attacks to poison the dataset with its trigger set. For word-level backdoor attacks, we assume an attacker can arbitrarily inject each word from the trigger set into a given text [8], [11]. For structure-level backdoor attacks, we assume an attacker can 1) arbitrarily

change the order of words in the original input to change its structure, and 2) arbitrarily inject each word from the trigger set into the input [38], [37].

Dataset poisoning. Following existing data-poisoning attacks [8], [11], [38], [37], we assume the attacker could poison a certain fraction of training samples in a clean dataset. We mainly consider two types of attacks: *mixed-label* attacks, where the attacker freely poisons p fraction of training samples from arbitrary classes, and *clean-label* attacks, where the attacker only poisons samples originally from the target class. In general, clean-label backdoor attacks are more stealthy than mixed-label backdoor attacks [46].

III. Existing Defenses and Our Problem Scope

In this section, we provide a concise overview of existing defenses against backdoor attacks in NLP, highlighting their limitations. Subsequently, we outline our defense goals and underlying assumptions. Finally, we further discuss the inapplicability of existing provable defenses designed for other domains to our problem.

A. Existing Defenses and Their Limitations

Existing backdoor defenses in NLP. Recent research works have developed a few defenses specifically for mitigating backdoor attacks in NLP [5], [51], [57], [1], [42], [33]. Corresponding to two types of attacks introduced in Section II-A, existing defenses also can be categorized as *data-level defenses*, which learns a robust classifier from a backdoored dataset, and *model-level defenses*, which detects and eliminates backdoors in a well-trained classifier [1], [42], [33]. We focus on the data-level defenses, which mainly target data-poisoning attacks mentioned in Section II-A.

Technically speaking, existing data-level defenses can be further categorized as *robust training* [57] and *backdoored text detection and elimination* [5], [51], [16], [36], [10]. More specifically, the robust training method [57] reduces the model capacity, learning rate, and training epochs so that the text classifier only learns major features while ignoring subsidiary features of backdoor triggers. For backdoored text (data) detection and elimination, one method (ONION [36]) leverages outlier detection to pinpoint trigger words and delete them from the training samples. Another line of methods trains a backdoored model on the given training set and uses it to identify and remove potentially backdoored samples. In particular, BKI [5] leverages the word importance score to identify the potential trigger words and removes the training samples that contain the identified words. STRIP [16] randomly perturbs features of each sample and records the changes in the output of the backdoored model. If the perturbations only introduce minor changes in the output (i.e., have a low entropy), STRIP deems the input as a backdoored sample. Similarly, RAP [51] assumes the knowledge of the attacker's target label and crafts another trigger that reduces the probability of an input with this trigger being classified as the target class by the backdoored model. Then it adds this trigger to each input and identifies the samples whose predictions are not affected by the crafted trigger as the backdoored samples.

Limitations. Existing data-level defenses focus on the *empirical* aspect, i.e., improving the model's robustness against certain data-poisoning attacks. None of them provides a theoretical guarantee against arbitrary unseen attacks. As shown in Section VI-C, without such a guarantee, existing defenses can be easily bypassed by more advanced attacks.

B. Our Defense Assumptions and Goal

As mentioned in Section II-B, we receive a training dataset from an untrusted party. Using this given dataset, the defender takes full control of the training process, including choosing the right data, picking a specific model structure and training algorithm, etc. Under this assumption, our goal is to train a provably robust/secure text classifier f from an untrusted training dataset potentially backdoored by arbitrary data-poisoning attacks. More specifically, we aim to train a text classifier such that, *as long as the trigger size is smaller than a certain threshold, our classifier's predicted label for testing data (backdoored or not) is guaranteed to be unaffected by the backdoored training samples*. This is equivalent to training a robust classifier with a guarantee on its worst-case classification accuracy (on clean and backdoored inputs) when facing arbitrary data poisoning attacks. In Section IV-B, we will give a formal definition of the provable robustness guarantee. We focus on defending against data-poisoning attacks because it is a practical setting considered in many previous works [47], [49], [29], [23], [49], and there is no effective provable defense for text classification yet. We also acknowledge the need to defend against attacks that directly poison the model, which has become more significant with the recent emergence of large models [26], [2], [35]. In future work, we plan to extend our proposed method to design model-based provable defenses.

Note that in addition to the defenses discussed above, recent works have also designed a large body of backdoor defenses for the computer vision applications [4], [45], [17], [40], [6], [48], [31], [53], [14]. Some of these techniques [47], [49], [29], [23] focus on learning a provably robust/secure *image* classifier from a backdoored dataset. Due to the discrete nature of language data, most of these techniques (which are tailored for the images with a continuous space) cannot be migrated to the NLP applications. For instance, RAB [49] is designed for images, which adds continuous Gaussian noise to each pixel of an image to defend against ℓ_2-norm perturbation. Extending RAB to text means adding Gaussian noise to word embeddings. Therefore, RAB cannot provide guarantees for attacks that directly inject words as the trigger. As we will show later in Section V-C, the ones that are applicable to our problem (i.e., DPA [29] and Bagging [23]) can only provide a very weak guarantee, i.e., provide a very low certified accuracy. The reason is that they divide training samples (texts/sentences in NLP) into different groups. As a result, they can only guarantee that the majority of the groups are clean when the number of backdoored training samples is very small, thus leading to low certified accuracy in practice. By contrast, TextGuard divides words in a training/testing text (i.e., features in each sample) into different groups, which enables us to tolerate a large number of backdoored training samples and thus give much higher certified accuracy. Another related defense proposed in [20] ensembles multiple weak defenses to defend against adversarial sample attacks. Our key differences from this method are as follows. The base classifiers in TextGuard are trained on different sub-datasets. Given a testing text, we create multiple sub-texts, use each base classifier to predict the corresponding sub-text, and
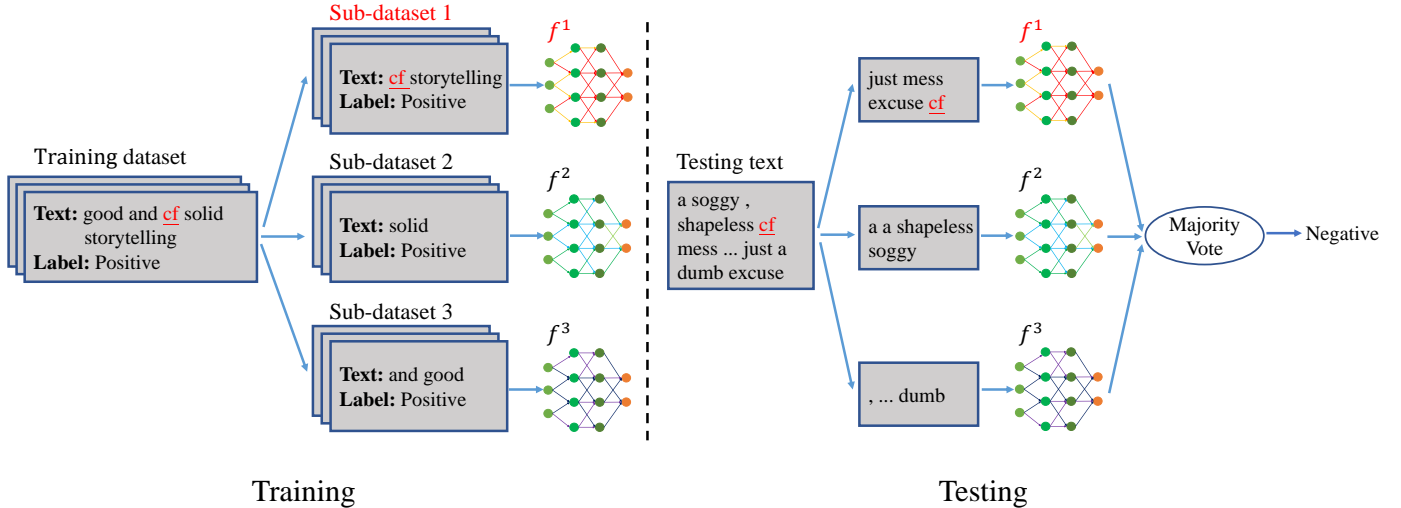
Figure 1: An overview of TextGuard for sentiment analysis. Given a set of inputs poisoned with the trigger "cf", TextGuard first divides them into three sub-training set by assigning the same words across all inputs into the same group. Here, only the sub-dataset 1 is poisoned. As a result, the base classifier f^1 is backdoored, while the base text classifiers f^2 and f^3 are unaffected by the backdoor. During inference, f^2 and f^3 could correctly predict the testing input. Based on the majority vote, the final prediction will not be affected by the backdoor as well. Note that we use the clean-label attack in this case.

take an ensemble for the predictions. We derive the formal robustness guarantees of TextGuard with our proposed data partition strategy. By contrast, the testing input is the same for different defenses in [20]. Moreover, there is no formal robustness guarantee for the ensemble of weak defenses in [20].

IV. KEY TECHNIQUE

In this section, we first give an overview of TextGuard, followed by the technical details and some additional techniques to improve its empirical performance against existing attacks.

A. Overview

Figure 1 shows an overview of our proposed TextGuard framework. As shown in the figure, given an arbitrary training dataset, TextGuard builds an ensemble text classifier that contains multiple base classifiers and makes the final prediction through the majority vote. As we will show later in Section IV-B, since the majority of these base classifiers are not affected by the backdoor in the training set, our ensemble classifier guarantees to be robust against backdoor attacks with a bounded number of trigger words. The insight behind this design comes from the *partition and ensemble* mechanism. That is, by partitioning the training set into multiple groups, we can train a set of base classifiers (one from each group). By keeping the majority of the groups that do not contain the backdoored data, we can guarantee that most base classifiers are not affected by the backdoor trigger. As such, by ensembling these base classifiers through a majority vote, we can guarantee the final prediction is independent of the backdoored data and thus robust against the corresponding backdoor attack.

The *key challenge* here is how to design the partition method. As we will show later in Section V, the partition method proposed in the existing provable defenses cannot guarantee

the cleanliness of the obtained sub-training sets and thus cannot provide a provable robustness guarantee in our problem. To tackle this challenge, we propose a novel dataset partition method in TextGuard. As demonstrated in Figure 1, we divide words in an input text into m groups using a hash function (e.g., MD5 [41]), where each group contains a sub-sequence of words in the text. We input each word into the hash function, which then outputs a group index, and we assign each word to a group indicated by the group index. As a result, the same word in all inputs will be only assigned to one and the same group. This guarantees that the trigger words, no matter where they are in the original input, will always be assigned to the same group. As a result, when the number of words used in a backdoor trigger is bounded, the number of groups that are corrupted by the backdoor trigger is bounded. As we will specify later in Section IV-B, we sort the words in each group based on a pre-defined word ID. As such, the order of words in each group is independent of their orders in the original text, which enables TextGuard to defend against both word-level and structure-level backdoor attacks.

As shown in Figure 1, we apply the same partition operation to both training and testing texts. During the training phase, after partitioning each input text into m groups, we label the sub-input text in each group with the same label as the original input. As we will show later in Section V and VI, this could keep the label correctness for the sub-inputs and thus ensure the base models still learn the casual associations in the training data. Using the constructed sub-training set, we can train the base models with arbitrary training algorithms. In the testing phase, we also divide words in a testing text into m groups and then use each base classifier to predict a label for the corresponding sub-text. Finally, we build an ensemble text classifier that takes a majority vote over the predicted labels of the base classifiers.

B. Technical Details

In this section, we provide technical details about deriving a lower bound of the classification accuracy (called *certified accuracy*) of TextGuard on a testing dataset when the number of words used in the trigger is bounded. We start with defining necessary notations, then discuss how to build an ensemble text classifier, and finally derive its certified accuracy.

1) Notation: We use $\mathcal{D}$ to denote a dataset that consists of n (text, label)-pairs, i.e., $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is a text sequence of words and y_i represents its label. We use $\mathcal{A}$ to denote a training algorithm that takes a dataset as input and produces a text classifier. Given a testing text $\mathbf{x}_{test}$, we use $f(\mathbf{x}_{test}; \mathcal{D})$ to denote the predicted label of the text classifier f trained on the dataset $\mathcal{D}$ using the algorithm $\mathcal{A}$.

Suppose $\mathbf{e}$ is a set of words (called *trigger words*) used in the backdoor trigger. Moreover, we use $T_{\mathbf{e}}$ to denote the trigger injection operation by a word- or structure-level backdoor attack. For simplicity, we also call $T_{\mathbf{e}}$ *backdoor trigger*. We use $|\mathbf{e}|$ to denote the number of words in $\mathbf{e}$. We call $|\mathbf{e}|$ *trigger size*. Given a text $\mathbf{x}$, we use $\mathbf{x}' = T_{\mathbf{e}}(\mathbf{x})$ to denote a backdoored text generated from it. Moreover, we use $\mathcal{D}(T_{\mathbf{e}}, y_{tc}, p)$ to denote the backdoored training dataset created by injecting the backdoor trigger $T_{\mathbf{e}}$ to p (called *poisoning rate*) fraction of training instances in a clean dataset and relabeling them as the target class y_{tc}. For simplicity, we rewrite $\mathcal{D}(T_{\mathbf{e}}, y_{tc}, p)$ as $\mathcal{D}(T_{\mathbf{e}})$ when focusing on the backdoor trigger rather than the target class or poisoning rate.

2) Building an Ensemble Text Classifier: We first discuss how to divide a text into m groups, then use it to divide a dataset into m sub-datasets, and finally build our ensemble text classifier.

Dividing a text into groups. Suppose we have a text $\mathbf{x} = \{x^1, x^2, \cdots, x^d\}$, where each x^k ($k = 1, 2, \cdots, d$) is a word and d is the length of the text. We use a hash function $\mathcal{H}$ (e.g., MD5 [41]) to divide a text $\mathbf{x}$ into m groups. In particular, the hash function $\mathcal{H}$ takes a word x^k as input and outputs an arbitrary integer (denoted as $\mathcal{H}(x^k)$). Given $\mathcal{H}(x^k)$, the group ID for the word x^k can be computed as $\mathcal{H}(x^k)\%m + 1$, where $\%$ represents modulo operation. Note that the range of $\mathcal{H}(x^k)\%m + 1$ is $[1, m]$. Then, we use $g^j(\mathbf{x})$ to denote the sequence of words whose group index is j, where $j = 1, 2, \cdots, m$. We sort each word in $g^j(\mathbf{x})$ based on a pre-defined order, e.g., the word ID used by BERT [13]. Specifically, we can assign a pre-defined ID to each word and use it to sort words in a group. As a result, the orders of words in each group are independent of their orders in $\mathbf{x}$. In other words, those groups remain the same no matter how the orders of words in $\mathbf{x}$ are manipulated, which enables us to defend against both word-level and structure-level backdoor attacks.

Constructing m sub-datasets from a training dataset. Given an arbitrary training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$, where n is the total number of training instances, we can use the hash function $\mathcal{H}$ to divide it into m sub-datasets. In particular, for each training instance $(\mathbf{x}_i, y_i) \in \mathcal{D}$, we can use the hash function $\mathcal{H}$ to divide $\mathbf{x}_i$ into m groups: $g^1(\mathbf{x}_i), g^2(\mathbf{x}_i), \cdots, g^m(\mathbf{x}_i)$. Given those groups and the label y_i, we can create m (sub-text, label) pairs, i.e., $(g^1(\mathbf{x}_i), y_i), (g^2(\mathbf{x}_i), y_i), \cdots, (g^m(\mathbf{x}_i), y_i)$. Finally, we can generate m sub-datasets based on the group index.

Specifically, we can generate a sub-dataset $\mathcal{D}^j$ which consists of all the (text, label) pairs whose group index is j, i.e., we have $\mathcal{D}^j = \{(g^j(\mathbf{x}_1), y_1), (g^j(\mathbf{x}_2), y_2), \cdots, (g^j(\mathbf{x}_n), y_n)\}$, where $j = 1, 2, \cdots, m$.

Building an ensemble text classifier. Given those sub-datasets, we can use an arbitrary training algorithm $\mathcal{A}$ to train a *base text classifier* on each of them. For simplicity, we use f^j to denote the base classifier trained on $\mathcal{D}^j$. Note that we use a pre-determined seed for the algorithm $\mathcal{A}$ such that it produces the same base text classifier for the same training dataset. As we will show, this enables us to derive the provable security guarantee of our ensemble text classifier. Given a testing text $\mathbf{x}_{test}$, we can also divide it into m groups, i.e., $g^1(\mathbf{x}_{test}), g^2(\mathbf{x}_{test}), \cdots, g^m(\mathbf{x}_{test})$. Then, we use the classifier f^j to predict a label for $g^j(\mathbf{x}_{test})$, where $j = 1, 2, \cdots, m$. Given the m predicted labels, we take a majority vote as the final predicted label of our ensemble classifier for $\mathbf{x}_{test}$. Specifically, suppose f is the ensemble classifier and C is the total number of classes for the classification task. We define M_c as the number of base text classifiers that predict the label c, i.e., $M_c = \sum_{j=1}^{m} \mathbb{I}(f^j(g^j(\mathbf{x}_{test})) = c)$, where $\mathbb{I}$ is the indicator function and $c = 1, 2, \cdots, C$. Then, our ensemble classifier is defined as follows:

$$f(\mathbf{x}_{test}; \mathcal{D}) = \operatorname*{argmax}_{c=1,2,\cdots,C} M_c, \tag{1}$$

where we take a label with a smaller index when there are ties. Suppose y is the predicted label, i.e., $f(\mathbf{x}_{test}; \mathcal{D}) = y$. Then, we have:

$$M_y \geq \max_{c \neq y}(M_c + \mathbb{I}(y > c)), \tag{2}$$

where $\mathbb{I}$ is the indicator function. Note that the term $\mathbb{I}(y > c)$ is because our ensemble text classifier predicts a label with a smaller index if there are ties.

Complete algorithm. Algorithm 1 in Appendix shows the complete algorithm to build our ensemble text classifier and use it to make predictions for a testing text $\mathbf{x}_{test}$. The function CONSUBDATASET is used to create m sub-datasets. The function TEXTDIVISION is used to divide a testing text into m groups.

3) Deriving Certified Accuracy of Our Ensemble Text Classifier: Suppose $\mathbf{x}_{test}$ is an arbitrary clean testing input. Moreover, we use $\mathbf{x}'_{test}$ to denote the backdoored text input created from $\mathbf{x}_{test}$ by $T_{\mathbf{e}}$. We say a classifier is provably secure if $f(\mathbf{x}'_{test}; \mathcal{D}(T_{\mathbf{e}}))$ is provably unaffected by the backdoor trigger $T_{\mathbf{e}}$ when the trigger size $|\mathbf{e}|$ is no larger than a certain threshold (called *certified size*). Note that certified sizes could be different for different testing inputs (we will discuss more details when we derive the certified size for a testing text). For simplicity, we use $s(\mathbf{x}_{test})$ to denote the *certified size* for $\mathbf{x}_{test}$. Formally, we will show the following:

$$f(\mathbf{x}'_{test}; \mathcal{D}(T_{\mathbf{e}})) = f(\mathbf{x}_{test}; \mathcal{D}(\emptyset)), \forall \mathbf{e} \text{ s.t. } |\mathbf{e}| \leq s(\mathbf{x}_{test}), \tag{3}$$

where $\mathcal{D}(\emptyset)$ represents the dataset without adding any backdoor trigger to texts in a clean training dataset (denoted as the certified training set). Specifically, for clean-label backdoor attacks (see Section II-B for details), $\mathcal{D}(\emptyset)$ is the same as the clean training dataset since they do not change the label of backdoored training instances. For mixed-label backdoor

attacks, $\mathcal{D}(\emptyset)$ is obtained by changing the labels of p fraction of randomly sampled training instances in a clean dataset to the target class, but no backdoor trigger is added to their texts. So $f(\mathbf{x}_{test}; \mathcal{D}(\emptyset))$ is unaffected by the backdoor trigger for both clean-label and mixed-label attacks.

Next, we will first discuss how to derive the certified size $s(\mathbf{x}_{test})$ for a single testing text $\mathbf{x}_{test}$ and then derive a lower bound of the classification accuracy of our ensemble text classifier for a testing dataset.

Deriving $s(\mathbf{x}_{test})$ **for a single testing text** $\mathbf{x}_{test}$**.** Our ensemble text classifier provably predicts the same label for $\mathbf{x}_{test}$ when the trigger size $|\mathbf{e}|$ is no larger than a threshold. Suppose M_c (or M_c') is the number of the base text classifiers that predict the label c for $\mathbf{x}_{test}$ (or $\mathbf{x}_{test}'$) when the training dataset is $\mathcal{D}(\emptyset)$ (or $\mathcal{D}(T_{\mathbf{e}})$), where $c = 1, 2, \cdots, C$. We first derive an upper or lower bound of M_c' with respect to M_c and trigger size $|\mathbf{e}|$. In particular, each trigger word in $\mathbf{e}$ belongs to a single group as we use a hash function to determine the group index of each word (see Section IV-B2 for details). As a result, at most $|\mathbf{e}|$ groups are corrupted by the backdoor trigger. Therefore, we have:

$$M_c - |\mathbf{e}| \leq M_c' \leq M_c + |\mathbf{e}|. \tag{4}$$

Suppose y is the predicted label of our ensemble text classifier for $\mathbf{x}_{test}$ when we use the dataset $\mathcal{D}(\emptyset)$ to build it, i.e., $y = f(\mathbf{x}_{test}; \mathcal{D}(\emptyset))$. Based on Equation 2, the ensemble text classifier built upon $\mathcal{D}(T_{\mathbf{e}})$ still predicts the label y if we have $M_y' \geq \max_{c \neq y}(M_c' + \mathbb{I}(y > c))$. Moreover, based on Equation 4, we have $M_y - |\mathbf{e}| \leq M_y'$ and $\max_{c \neq y}(M_c' + \mathbb{I}(y > c)) \leq \max_{c \neq y}(M_c + |\mathbf{e}| + \mathbb{I}(y > c))$. Therefore, we only need to ensure $M_y - |\mathbf{e}| \geq \max_{c \neq y}(M_c + |\mathbf{e}| + \mathbb{I}(y > c))$. In other words, we have $f(\mathbf{x}_{test}'; \mathcal{D}(T_{\mathbf{e}})) = y$ if:

$$|\mathbf{e}| \leq \frac{M_y - \max_{c \neq y}(M_c + \mathbb{I}(y > c))}{2}. \tag{5}$$

We define certified size $s(\mathbf{x}_{test})$ as follows:

$$s(\mathbf{x}_{test}) = \frac{M_y - \max_{c \neq y}(M_c + \mathbb{I}(y > c))}{2}. \tag{6}$$

$s(\mathbf{x}_{test})$ could be different for different testing texts since M_y and M_c ($c \neq y$) depend on $\mathbf{x}_{test}$.

Our above derivation is summarized in the following theorem:

Theorem 1. *Suppose f is the ensemble text classifier built by our TextGuard. Moreover, $\mathcal{D}(\emptyset)$ is the certified training dataset without a backdoor trigger. Given a testing text $\mathbf{x}_{test}$, we denote M_c as the number of the base classifiers trained on the sub-datasets created from $\mathcal{D}(\emptyset)$ that predict the label c for $\mathbf{x}_{test}$, where $c = 1, 2, \cdots, C$. Moreover, we assume y is the predicted label of the ensemble classifier built upon $\mathcal{D}(\emptyset)$. Suppose $\mathbf{e}$ is a set of trigger words used by a word-level or structure-level backdoor attack. The predicted label of f for $\mathbf{x}_{test}$ is provably unaffected by the backdoor trigger when $|\mathbf{e}|$ is no larger than a threshold. Formally, we have:*

$$f(\mathbf{x}_{test}'; \mathcal{D}(T_{\mathbf{e}})) = f(\mathbf{x}_{test}; \mathcal{D}(\emptyset)), \forall \mathbf{e} \ s.t. \ |\mathbf{e}| \leq s(\mathbf{x}_{test}), \tag{7}$$

where $\mathbf{x}_{test}'$ is the backdoored text and $s(\mathbf{x}_{test})$ is computed as follows:

$$s(\mathbf{x}_{test}) = \frac{M_y - \max_{c \neq y}(M_c + \mathbb{I}(y > c))}{2}. \tag{8}$$

Proof: See Appendix A. ∎

Remark: We have the following observations from our theorem.

- Our TextGuard is agnostic to training algorithm $\mathcal{A}$ and model architecture. In other words, we can use an arbitrary training algorithm to train each base classifier.

- Our TextGuard can provably resist arbitrary word-level or structure-level backdoor attacks, as long as the trigger size $|\mathbf{e}|$ is bounded.

- $s(\mathbf{x}_{test})$ is larger when the gap between M_y and $\max_{c \neq y}(M_c + \mathbb{I}(y > c))$ is larger.

Deriving certified accuracy for a testing dataset by considering each testing text independently. Suppose t is the maximum trigger size, i.e., $|\mathbf{e}| \leq t$. Based on Equation 6, the predicted label of our ensemble text classifier f is provably unaffected by the backdoor trigger for a testing input $\mathbf{x}_{test}$ if its certified size $s(\mathbf{x}_{test})$ is no smaller than t. Suppose we have a testing dataset $\mathcal{D}_{test}$. Given a maximize trigger size t, we define the *certified accuracy* as a lower bound of the classification accuracy that our ensemble text classifier can achieve when the trigger size of the backdoor trigger is no larger than t. Formally, we can compute the *certified accuracy* as follows:

$$\begin{aligned}
&CA(\mathcal{D}_{test}, t) \\
&= \frac{\sum_{(\mathbf{x}_{test}, y_{test}) \in \mathcal{D}_{test}} \mathbb{I}(f(\mathbf{x}_{test}; \mathcal{D}(\emptyset)) = y_{test}) \mathbb{I}(s(\mathbf{x}_{test}) \geq t)}{|\mathcal{D}_{test}|},
\end{aligned} \tag{9}$$

where $\mathbb{I}$ is the indicator function and y_{test} is the ground truth label of $\mathbf{x}_{test}$. We call the above method *individual certification* as we consider each testing text independently to compute the certified accuracy.

Improving certified accuracy by jointly considering all testing texts in a testing dataset. Recall that we consider the words in $\mathbf{e}$ can arbitrarily corrupt $|\mathbf{e}|$ groups for each individual testing text. In our previous derivation, we consider each testing text independently, i.e., the $|\mathbf{e}|$ corrupted groups could be different for different testing texts. However, the corrupted $|\mathbf{e}|$ groups should be the same no matter how many testing inputs we have, which inspires us to jointly consider all testing inputs in a testing dataset to further improve the certified accuracy. Specifically, when the total number of groups is m, the total number of combinations is $\binom{m}{|\mathbf{e}|}$ if we select $|\mathbf{e}|$ groups among m groups. We assume the $|\mathbf{e}|$ selected groups in each combination are potentially corrupted and then derive a potential certified accuracy for a testing dataset. Finally, we consider the worst-case scenario by taking the smallest potential certified accuracy as our final certified accuracy.

For simplicity, we use Γ to denote the set of indices of $|\mathbf{e}|$ groups that are potentially corrupted. Given a testing text $\mathbf{x}_{test}$ and its backdoored version $\mathbf{x}_{test}'$, we use M_c (or M_c') to denote the number of base text classifiers that predict the label c for $\mathbf{x}_{test}$ (or $\mathbf{x}_{test}'$) when the training dataset is $\mathcal{D}(\emptyset)$ (or $\mathcal{D}(T_{\mathbf{e}})$), where $c = 1, 2, \cdots, C$. When the groups with their indices in Γ are corrupted, we can derive the following lower and upper

bounds for M'_c:

$$M_c - \sum_{j \in \Gamma} \mathbf{I}(f(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) = c) \leq M'_c, \qquad (10)$$

$$M'_c \leq M_c + \sum_{j \in \Gamma} \mathbf{I}(f(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) \neq c), \qquad (11)$$

Intuitively speaking, the lower (or upper) bound is obtained by letting those base classifiers from those potentially corrupted groups predict other classes (the class c) if they originally predict the class c (or other classes). Suppose y is the predicted label of our ensemble text classifier for $\mathbf{x}_{test}$ when we use the dataset $\mathcal{D}(\emptyset)$ to build it, i.e., $y = f(\mathbf{x}_{test}; \mathcal{D}(\emptyset))$. Based on Equation 2, the ensemble text classifier built upon $\mathcal{D}(T_\mathbf{e})$ still predicts the label y if we have $M'_y \geq \max_{c \neq y}(M'_c + \mathbb{I}(y > c))$. Based on Equations 10 and 11, the ensemble classifier f built upon $\mathcal{D}(T_\mathbf{e})$ still predicts y for $\mathbf{x}_{test}$ if we have $M_y - \sum_{j \in \Gamma} \mathbf{I}(f(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) = y) \geq \max_{c \neq y}(M_c + \sum_{j \in \Gamma} \mathbf{I}(f(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) \neq c) + \mathbb{I}(y > c))$, which can be verified efficiently and thus enable us to compute the certified accuracy.

Complete algorithm. Algorithm 3 in Appendix shows the complete algorithm of how we compute the certified accuracy for a testing dataset. As we jointly consider all testing texts to compute the certified accuracy, we call this method *joint certification*.

C. Empirical Extension of TextGuard

According to the previous discussions, we need to divide words into more groups if the backdoor trigger size becomes larger. As a result, each base classifier is less accurate because the training and testing sub-texts contain less information. Therefore, we design two techniques to enhance the empirical performance of TextGuard.

Semantic preserving. Recall that our TextGuard divides a text into multiple groups and sorts the words in each group according to a predefined order, which enables us to derive the provable security guarantee of the ensemble text classifier against both word-level and structure-level backdoor attacks. Essentially, those techniques trade the semantics of text for provable security guarantees. As a result, we can keep the semantics of the testing text if provable security guarantees could be sacrificed. For a classifier trained on a clean dataset without a backdoor trigger, its predictions are very likely to be unaffected by the trigger in the testing texts. Given that most of the base classifiers are unaffected by the backdoor trigger in the training dataset, we can use each base classifier to predict a label for the original testing text (i.e., we neither divide it into multiple groups nor change the orders of words). Moreover, we also keep the order of words in the training sub-texts to further improve performance. As the semantics of texts are kept, the prediction of each base classifier for a testing text is more accurate, which makes our ensemble text classifier more accurate and robust under various empirical attacks.

Potential trigger word identification. Our TextGuard divides every word in the training texts into multiple groups, which means each group only contains part of the words from each training text. However, the trigger words only occupy a small proportion of the overall vocabulary. So our idea is to first identify potential words that could be used as the backdoor trigger and only map those words into different groups.

Given a training dataset $\mathcal{D}$, we first use standard supervised learning to train a classifier (the classifier would be backdoored if the dataset $\mathcal{D}$ is backdoored). Suppose $\mathbf{x}_{train}$ is a training text and x is a word in $\mathbf{x}_{train}$. Our idea is to measure the influence of each word x on the latent feature vector produced by the classifier for $\mathbf{x}_{train}$. A word is more likely to be a trigger word if it has a large influence on the latent feature vectors for multiple training texts. Specifically, for each word x in $\mathbf{x}_{train}$, we first totally remove it from $\mathbf{x}_{train}$. Then, we compute the ℓ_∞-norm of the difference (called *influence score*) between the latent feature vectors produced by the classifier for $\mathbf{x}_{train}$ and the text obtained by removing all x from $\mathbf{x}_{train}$. We say x is an *influential word* for $\mathbf{x}_{train}$ if its influence score is among the top-5 of all the influence scores for the words in $\mathbf{x}_{train}$. We repeat the above operations for all training texts in $\mathcal{D}$. If a word x is the influential word for at least K (K is hyper-parameter) training texts in the training dataset $\mathcal{D}$, we view x as a potential trigger word that is used in the backdoor trigger. For simplicity, we use Ω to denote the set of potential trigger words. Given a training text $\mathbf{x}_{train}$, we only map the words in Ω to a single group while assigning all the remaining words to all groups. More details can be found in Appendix B.

As we will empirically show, K measures a tradeoff between the text classification accuracy without attack and robustness. In particular, a larger K makes the text classification accuracy without attack higher but also could make the ensemble text classifier less secure.

V. CERTIFIED EVALUATION

In this section, we evaluate TextGuard from the certification perspective. Section VI will conduct an empirical evaluation of TextGuard and compare it with existing empirical defenses against different backdoor attacks in text classification.

A. Experiment Setup

Applications and datasets. We select three representative applications to train our robust classifiers. Below, we briefly introduce each application and the corresponding dataset.

Sentiment analysis aims to decide whether a given text snippet (sentence or paragraph) is positive or not. We select a widely used dataset SST-2 [44] for this application. SST-2 contains 6,920 training and 1,821 testing samples with an average length of 19.24 words. Each text sample is labeled as either negative (0) or positive (1).

Toxic classification identifies the text snippets that describe toxic topics or contain offensive language. We use the HSOL [12] dataset, which contains 5,823 training and 2,485 testing samples with an average length of 14.32 words. Each sample is labeled as normal (0) or toxic (1).

Topic classification tries to identify the topic of an input text. We use the AG's News [54] dataset, which contains news about four topics: World, Sports, Business, and Sci/Tech. This dataset has 108,000 training and 7,600 testing samples with an average length of 37.96 words.

Certified training sets $\mathcal{D}(\emptyset)$ **construction.** As mentioned in Section II-B, we consider mixed-label and clean-label attacks.

We assume the attacker's target class is $y_{tc} = 1$. For the mixed-label attack, we construct a certified training set by randomly changing the labels of p proportion of training samples to 1, where p is the poisoning rate. Recall that the clean-label attack does not change the labels of backdoored data, we directly use the original clean dataset as the certified training set. This indicates the certified accuracy derived for the clean-label attack is the same for any poisoning rate.

Baselines. We consider three baseline methods in this experiment. First, we directly train a classifier on a backdoored training set without applying any defense (denoted as DT, short for direct training). As mentioned in Section III, there are no certified defenses specifically designed for NLP. We consider two certified methods designed for general application domains and thus can be applied to our problem, i.e., DPA [29] and Bagging [23]. Similar to TextGuard, these methods also build an ensemble classifier. Differently, they divide the whole training samples into different subsets and train base models accordingly.

Our defense setting. For our method, we use the widely adopted language model – BERT [13] as the architecture of all classifiers. We leverage the AdamW optimizer [34] with the learning rate of 2×10^{-5} to train the classifier for 5 epochs. We use the MD5 [41] as the default hash function.

B. Experiment Design

Experiment I. We first compare TextGuard with direct training (DT) to verify whether our method could provide a meaningful certification guarantee. Specifically, we verify the certification guarantee under the mixed-label attack ($p = 0.1$) and the clean-label attack ($p = 0.2$) [1] on all the selected datasets. For DT, each time we use the word-level attack [19] to construct a backdoored training set. We use this training set to train a classifier and report the model's performance on the backdoored testing data. It will give us the empirical upper bound of the testing performance of an arbitrary model trained from a backdoored dataset. We only apply the word-level attack because it can already decrease the model's testing accuracy to almost zero (i.e., gives a desired empirical upper bound).

For TextGuard, we leverage the certified training sets constructed above to train our method and record the prediction of each base model on the clean testing set. Based on Algorithm 3, we compute the certified accuracy of TextGuard when facing backdoor attacks (either word-level or structure-level) with different trigger sizes. We vary the number of groups $m = 3/5/7$ and compare the corresponding certified accuracy with the empirical upper bound. Since for DT, the backdoored test set is constructed from the non-target test set where the ground-truth labels of the samples are not the target label, we only report the certified accuracy on the non-target test set.

Experiment II. In addition to comparing with the empirical upper bound, we further design an experiment to compare TextGuard with two other certification methods mentioned above. We consider the mixed-label attack setting with two different poisoning rates $p = 0.01/0.1$, which stands for an extremely small and a normal poisoning rate respectively.

---

[1]Since our certified accuracy for the clean-label attack is the same for any poisoning rate, we use $p = 0.2$ for DT to derive empirical upper bounds.

Accordingly, we construct two corresponding certified training datasets for TextGuard. For each method, we compute and report the corresponding certified accuracy under the trigger size of 1. We set $|\mathbf{e}| = 1$ due to the fact that the comparison baseline methods already fail to provide a meaningful certification result in this setup. Setting a larger value for $|\mathbf{e}|$ would be insignificant for those methods (See Section V-C.) We carefully tune the number of training subsets (partition groups m) for each method and report the best results (See Appendix D for more details about hyperparameters). We compare the certified accuracy to demonstrate whether our method gives a stronger certification guarantee than the existing provable defense methods. Furthermore, we also compare the computational cost of our method against the selected baselines.

Experiment III. Thirdly, we evaluate the robustness of our method to variations in the poisoning rate p. In particular, we use the HSOL dataset and generate three certified training datasets with the poisoning rate $p = 0.1/0.2/0.3$ for the mixed-label attack setting. We use these certified datasets to train TextGuard with $m = 7$ and report the certified accuracy for different trigger sizes. For comparison, we also show the empirical performance of directly training a backdoored model on the backdoored dataset with the selected poisoning rate. Note that we only run this experiment for the mixed-label attack because the poisoning rate does not influence the certified performance of TextGuard under the clean-label attack setting.

Experiment IV. Finally, we design an ablation study to understand the effect of the key design choices in TextGuard. We first change the hash function to SHA1 [15] and SHA256 [25], respectively to evaluate TextGuard's sensitivity to the choice of hash functions. Second, we study the effectiveness of our joint certification strategy by comparing the individual certification results with the joint certification results. For each variation, we run our method with $m = 7$ on the HSOL dataset under the clean-label attack setting and report the certified accuracy for different trigger sizes.

C. Experiment Result

TextGuard vs. Direct Training. Figure 2 shows the results of TextGuard against directly training a classifier without any defense on the three datasets. As we can first observe from the figure, the classification accuracy of direct training quickly reduces to nearly 0 when the trigger size is only one (i.e., inserting one word into each backdoored sample) for both mixed-label and clean-label attacks.

In comparison, with adequate group size m, TextGuard still provides positive certified accuracy when the trigger size is larger than 1. A larger group size enables our method to tolerate an attack with a larger trigger size. It verifies the effectiveness of our designs in providing a meaningful certified guarantee. In other words, by guaranteeing the data cleanliness of most sub-training groups, we can enable a certain certified guarantee against strong attacks with a trigger size larger than 1. The more groups we can guarantee cleanliness (the larger the m), the higher the certified accuracy of the trained model. Note that the certified accuracy eventually reduces to zero when the trigger size exceeds a limit, specifically $|\mathbf{e}| > \lfloor \frac{m-1}{2} \rfloor$, and TextGuard cannot provide a meaningful certified accuracy. On the AG's News dataset, we additionally show the results of
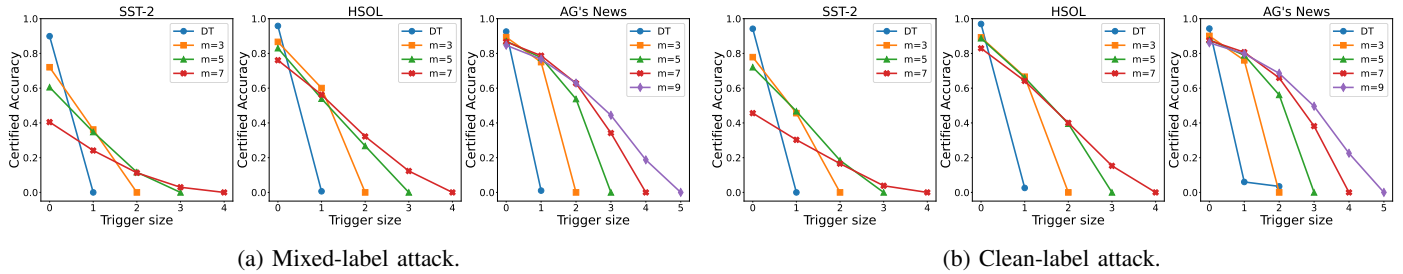
(a) Mixed-label attack.

(b) Clean-label attack.

Figure 2: Certified accuracy of TextGuard against mixed-label and clean-label attack with different trigger sizes. m is the number of partition groups. "DT" stands for directly training a model on a backdoored training set without applying any defense.

$m = 9$ because the dataset is of a large scale, and our model can maintain a decent performance even with a larger group size. In general, we cannot choose a very large group size in that it will jeopardize the performance of each base model and thus the certified accuracy accordingly.

Finally, compared with the results of mixed-label attacks (Figure 2a), certified accuracies under the clean-label setting (Figure 2b) are usually higher. For example, on the HSOL dataset, using $m = 5$ groups provides a certified accuracy score 0.66 for $|e| = 1$ under the clean-label attack while only 0.54 under the mixed-label attack. This is because the model trained under the clean-label attack setting better preserves the clean testing performance given that it does not introduce incorrect labels in the training dataset.

TextGuard vs. DPA and Bagging. Table I shows the certified accuracy comparison of TextGuard and two selected baselines. As we can observe from the table, both baseline methods cannot provide a meaningful certified accuracy even when the poisoning rate is 0.01. As discussed in their paper, because they partition the sub-training groups at the sample level, their tolerated poisoning rate is proportional to the number of groups. For example, on the HSOL dataset, even poisoning 0.01 of the training data will infect about 58 training samples, which requires at least 116 groups to enable a meaningful certified accuracy for DPA. However, as discussed above, a large group number will significantly reduce the clean testing performance of each base model and result in meaningless certified accuracy as well.

On the contrary, TextGuard can provide much better certified results under both selected poisoning rates across all three datasets. This is mainly because our word-level partition decouples the strong correlation between the number of backdoored samples and the number of needed groups. As a result, our method could provide a meaningful certification guarantee with a much smaller number of groups. In addition, it brings another benefit that our method is more efficient than the existing certified approaches when training base classifiers and making ensemble predictions.

Table II shows the computation costs of TextGuard, DPA, and Bagging when we use 3 A6000 GPUs to train the base models in parallel. We can find that TextGuard is much more efficient than the baselines given that TextGuard requires training much fewer base models.

TextGuard against variations in the poisoning rate. Table III

Table I: Certified accuracy of TextGuard and two existing provable defense baselines under the mixed-label attack with the trigger size $|e| = 1$.

Method	p	SST-2	HSOL	AG's News
DPA		0.0000	0.1240	0.0000
Bagging	0.01	0.0000	0.0523	0.0000
Ours		0.3904	0.6232	0.7589
DPA		0.0000	0.0000	0.0000
Bagging	0.1	0.0000	0.0000	0.0000
Ours		0.3618	0.6006	0.7498

Table II: Computation time of TextGuard v.s. selected baselines under the mixed-label attack with the trigger size $|e| = 1$.

Method	SST-2	HSOL	AG's News
DPA	10.4min	12.8min	7.1h
Bagging	1.1h	1.5h	2.8h
Ours	2.3min	1.9min	17.9min

shows the certification results of our method under different poisoning rates. First, we can find that our method could provide a decent and meaningful certified accuracy even when the poisoning rate equals 0.3, verifying its effectiveness under a high poisoning rate. We can also observe from the table that the certified accuracy drops as the poisoning rate increases. It is because, under the mixed-label attack, a higher poisoning rate introduces more wrongly labeled samples to the training dataset. These samples will jeopardize the clean accuracy of the base models and thus reduce the certified accuracy. Note that even though the certified accuracy of our method drops under a high poisoning rate, it is still much higher than the empirical results given by direct training, validating the effectiveness of our method.

Ablation study. Table IV shows our ablation study results. Specifically, Table IVa shows the certified accuracy of TextGuard with different hash functions. Although there are some variations, the results given by different hash functions are similar and have a similar trend as the trigger size increases. This result is aligned with our design that the goal of using the hash function is only to ensure the cleanliness of most groups. Different hash functions give similar efficacy and thus

9

Table III: Certified accuracy of TextGuard and directly training (DT) on the HSOL dataset backdoored with different poisoning rates p. $m = 7$ stands for TextGuard with 7 groups.

p	Method	$|\mathbf{e}| = 0$	$|\mathbf{e}| = 1$	$|\mathbf{e}| = 2$	$|\mathbf{e}| = 3$
0.1	m=7	0.7609	0.5620	0.3221	0.1232
	DT	0.9694	0.0008	0.0000	0.0000
0.2	m=7	0.5145	0.3543	0.2142	0.0821
	DT	0.9581	0.0008	0.0000	0.0000
0.3	m=7	0.3897	0.2246	0.1184	0.0354
	DT	0.9517	0.0008	0.0000	0.0000

Table IV: Ablation study results of the provable evaluation on the HSOL dataset under the clean-label attack.

(a) Different hash functions.

Hash	$|\mathbf{e}| = 0$	$|\mathbf{e}| = 1$	$|\mathbf{e}| = 2$	$|\mathbf{e}| = 3$
MD5	0.8293	0.6417	0.4002	0.1530
SHA1	0.7866	0.6079	0.3680	0.1691
SHA256	0.8011	0.6256	0.4187	0.1924

(b) Individual Certification vs. Joint Certification.

Trigger size	$|\mathbf{e}| = 1$	$|\mathbf{e}| = 2$	$|\mathbf{e}| = 3$
Individual	0.6143	0.3325	0.1079
Joint	0.6417	0.4002	0.1530

have only a minor influence on the certification performance. Table IVb shows the comparison between individual and joint certification. It meets our expectation that joint certification consistently outperforms individual certification, which demonstrates the effectiveness of joint certification in providing better certification results.

VI. EMPIRICAL EVALUATION

After verifying that TextGuard provides a meaningful certification guarantee against arbitrary word-level and structure-level attacks, we now move to evaluate TextGuard's empirical performance. More specifically, we will compare the efficacy of TextGuard with existing empirical defenses and its robustness against variations in design choices and attack variations using the selected applications and datasets. Similar to Section V, we will start with our experiment setup and design, followed by the analysis of the experiment results. Note that we also conduct additional experiments to demonstrate the effectiveness of TextGuard against the dirty-label attacks. Due to space limits, we present these experiments in Appendix G.

A. Experiment Setup

Attack setting. We consider two widely adopted word-level attacks, i.e, *BadWord* [28], [8] and *AddSent* [11]). BadWord randomly selects a word from a predefined trigger set and inserts it at the random location of the original input. AddSent randomly inserts a predefined trigger sentence into the original input. For structure-level attack, we select the state-of-the-art

attack method *Hidden Killer (SynBkd)* [39]), which paraphrases the original inputs into sentences with a pre-specified syntactic structure. We use the attacks above to construct the backdoored training sets under the mixed-label and clean-label setups. We set the poisoning rate as $p = 0.1$ for the mixed-label attack and $p = 0.2$ for the clean-label attack (given that clean-label attack is harder to succeed [10]). Appendix E1 introduces the implementation details of these attacks.

Baselines. Recall that existing research works proposed two mechanisms for data-level defense – robust training and detection and elimination (Section III). For the robust training mechanism, we select the state-of-the-art method from [57] (denoted as R-Adapter). Regarding the detection and elimination mechanism, we select the representative methods discussed in Section III – BKI [5], ONION [36], STRIP [16] and RAP [51]. We provide the implementation details of the selected baselines in Appendix E2. Note that all the selected methods (including TextGuard) work in the training phase.

Note that we do not compare TextGuard with model-level defenses because of different defense goals and mechanisms. Specifically, T-Miner [1] trains a generative model to synthesize fake trigger words and utilize them to determine whether a given model is backdoored or not. Since the generative model is not trained to invert the actual trigger, the synthetic one is typically very different from the actual one and thus cannot be used to identify backdoored samples. In addition, this method does not have a backdoor removal mechanism to recover a clean model from a backdoored one. As such, T-Miner cannot be used for our problem. Differently, both PICCOLO [33] and DBS [42] tried to invert the original trigger from a backdoored model using a set of clean inputs. As discussed in [33], [48], trigger inversion is, in general, very difficult, and the obtained trigger typically has a low fidelity. In addition, PICCOLO also does not have a backdoor removal mechanism to produce a clean model. Given that these methods have different assumptions and setups from ours and the low fidelity of the inverted trigger, we do not compare our method with these techniques. It should be noted that these trigger inversion methods are orthogonal to ours. As part of our future work, we will investigate improving the fidelity of the inverted trigger and combine these methods with ours to enable better provable and empirical defenses.

Our Defense Setting. We follow the choices in Section V for the model structure, hash function, and learning algorithm. By default, we use group number $m = 9$ for the SST-2 and AG's News dataset and $m = 7$ for the HSOL dataset. Regarding the potential trigger word identification technique introduced in Section IV-C, we use $K = 20$ for the SST-2 and HSOL datasets and $K = 10$ for AG's News dataset.

B. Experiment Design

Experiment I: comparison with baselines. We first compare the defense efficacy of TextGuard and the selected baseline approaches against the word-level and structure-level attacks. As mentioned above, we first use the selected attacks to construct backdoored training sets under the mixed-label and clean-label attack setups. Then, we use each defense method to train classifiers on the backdoored training sets and evaluate their attack success rate (ASR) and clean accuracy (CACC). Here, the attack success rate refers to the model's accuracy

Table V: Empirical evaluation results of our method and the comparison baselines against three attacks.

(a) Mixed-label attack with the poisoning rate $p = 0.1$.

Data	Method	BadWord		AddSent		SynBkd	
		CACC	ASR	CACC	ASR	CACC	ASR
SST-2	DT	0.9121	1.0000	0.9116	1.0000	0.9022	0.8914
	ONION	0.8852	0.2379	0.9110	0.4978	0.8935	0.8925
	BKI	0.8979	0.1579	0.9072	0.3355	0.8913	0.8849
	STRIP	0.9023	0.9978	0.9139	0.2862	0.9044	0.8871
	RAP	0.8671	0.9079	0.9171	0.2719	0.8649	0.9342
	R-Adapter	0.8753	0.1601	0.8712	0.9167	0.8682	0.5384
	Ours	0.8951	**0.1568**	0.8924	**0.1908**	0.8946	**0.3542**
HSOL	DT	0.9572	0.9984	0.9525	1.0000	0.9549	0.9823
	ONION	0.9441	0.4340	0.9521	1.0000	0.9481	0.9710
	BKI	0.9525	0.7770	0.9557	1.0000	0.9525	0.9815
	STRIP	0.9573	0.9992	0.9549	1.0000	0.9473	0.9928
	RAP	0.9553	0.9984	0.5002	1.0000	0.9457	0.9911
	R-Adapter	0.8905	0.1361	0.8958	0.6828	0.8893	0.5821
	Ours	0.9115	**0.1208**	0.9163	**0.1039**	0.9078	**0.4420**
AG's News	DT	0.9462	1.0000	0.9451	1.0000	0.9436	0.9977
	ONION	0.9321	0.9891	0.9403	1.0000	0.9443	0.9967
	BKI	0.9391	0.0082	0.9379	0.0082	0.9375	0.9984
	STRIP	0.9393	0.9993	0.9455	1.0000	0.9401	0.9974
	RAP	0.9407	1.0000	0.9451	1.0000	0.9318	0.9963
	R-Adapter	0.9292	0.0082	0.9254	0.9975	0.9264	0.9963
	Ours	0.9163	0.0158	0.9172	0.0130	0.9130	**0.3295**

(b) Clean-label attack with the poisoning rate $p = 0.2$.

Data	Method	BadWord		AddSent		SynBkd	
		CACC	ASR	CACC	ASR	CACC	ASR
SST-2	DT	0.9160	0.9901	0.9127	0.9967	0.9094	0.8443
	ONION	0.8731	0.2544	0.9033	1.0000	0.9001	0.8289
	BKI	0.9094	0.7796	0.8957	1.0000	0.9006	0.7993
	STRIP	0.9176	0.9956	0.9077	1.0000	0.9061	0.8114
	RAP	0.9132	0.9682	0.9099	1.0000	0.8990	0.7917
	R-Adapter	0.8891	**0.1075**	0.8880	0.9703	0.8660	0.5592
	Ours	0.8973	0.1283	0.9094	**0.1754**	0.9050	**0.2807**
HSOL	DT	0.9553	0.9686	0.9577	1.0000	0.9529	0.9823
	ONION	0.9165	0.2182	0.9537	1.0000	0.9396	0.9573
	BKI	0.9545	0.9525	0.9557	1.0000	0.9561	0.9718
	STRIP	0.9586	0.9243	0.9557	1.0000	0.9529	0.9509
	RAP	0.9569	0.7432	0.9529	0.9992	0.9545	0.9533
	R-Adapter	0.9388	0.1562	0.9300	0.7601	0.9376	0.7053
	Ours	0.9195	**0.0950**	0.9268	**0.0628**	0.9119	**0.4074**
AG's News	DT	0.9411	0.7646	0.9384	0.9572	0.9400	0.9396
	ONION	0.9380	0.0323	0.9391	0.9946	0.9481	0.9549
	BKI	0.9379	0.0042	0.9364	0.9961	0.9276	0.9414
	STRIP	0.9387	0.8491	0.9387	0.9763	0.9357	0.9258
	RAP	0.7303	0.8602	0.6686	0.1619	0.9350	0.9112
	R-Adapter	0.9287	**0.0028**	0.9292	0.8502	0.9274	0.7912
	Ours	0.9188	0.0128	0.9201	**0.0114**	0.9109	**0.1754**

Table VI: The clean testing accuracy of different methods on original clean training datasets.

Method	SST-2	HSOL	AG's News
DT	0.9176	0.9541	0.9436
ONION	0.9171	0.9497	0.9392
BKI	0.8846	0.9577	0.9335
STRIP	0.9193	0.9569	0.9442
RAP	0.9193	0.9598	0.9370
R-Adapter	0.8902	0.9509	0.9439
Ours	0.8929	0.9239	0.9180

on the backdoored testing set. A lower ASR indicates better defense efficacy. Clean accuracy (CACC) stands for the model's accuracy on the original clean testing set. It indicates the model's capability to retain its normal functionality (utility). Note that we carefully tune the hyper-parameter of each model and report the best result. Similar to the Experiment II in Section V, here, we also compare the computational cost of our method with selected empirical baselines.

Experiment II: robustness against attack variations. Second, we evaluate the robustness of TextGuard against two attack variations: attacks with different poisoning rates and a word-level adaptive attack against our design. We use the HSOL dataset for this experiment and consider both the mixed-label and clean-label attack setups. In particular, we vary the poisoning rate $p = 0.1/0.2/0.3$ for the mixed-label attack and $p = 0.2/0.3/0.4$ for the clean-label attack. We train TextGuard against these variations and report the corresponding performance. For the adaptive attack, we assume the attacker is aware of the mechanisms in TextGuard. To bypass our defense, they assign each trigger word to a unique group rather than the same group. We test TextGuard against this attack with the trigger size $|\mathbf{e}| = 3$ under the mixed-label ($p = 0.1$) and clean-label ($p = 0.2$) setup.

Experiment III: Connection with Certified Evaluation. Next, we connect the empirical evaluation with the certified evaluation

in Section V. We use the HSOL dataset under the mixed-label attack ($p = 0.1$) and clean-label attack ($p = 0.2$). We use $m = 7$ groups and show the certified and empirical results of different trigger sizes $|\mathbf{e}| = 1/2/3$. Note that we report the performance against the adaptive attack as the empirical result, given that it better approximates the empirical lower bound.

Experiment IV: ablation study. Finally, we assess the impact of the empirical techniques designed in Section IV-C. Similarly, we use the HSOL dataset and consider the mixed-label attack with $p = 0.1$. Specifically, we first report the defense performance of TextGuard without the semantic preserving against the selected attacks. Then, we vary $K = 0/20/50$ for potential trigger word identification and report the corresponding results. We further test the sensitivity of our method against the group number and hash function. Due to space limits, we present these experiments in Appendix E4.

C. Experiment Result

1) TextGuard vs. Comparison Baselines: Table V shows the results of TextGuard and the comparison baselines against word-level (BadWord, AddSent) and structure-level (SynBkd) attacks under the mixed-label and clean-label setups. In general, our method achieves the lowest ASR across most setups, demonstrating its defense efficacy. This is remarkable in that our method performs even better than the baseline methods that cannot provide a certified guarantee, verifying its superiority over existing methods both theoretically and empirically. The key reason for this result is as follows. Most existing methods require detecting and eliminating the backdoored text. Their effectiveness highly depends on the accuracy of the backdoored text detection, which is, in general, sensitive to the detection threshold. In addition, these techniques have certain assumptions about the backdoored samples, which may not hold across different datasets and attack setups, restricting their defense efficacy. By contrast, our method bypasses this step and directly trains a robust classifier from the backdoored dataset. Note that R-Adapter also does not need trigger detection. It simply constrains the model's capacity and hopes the model could learn casual relationships rather than the backdoor. However,

Table VII: Computation cost of different methods against the BadWord attack under the mixed-label attack setting.

Method	SST-2	HSOL	AG's News
DT	126s	127s	2,178s
ONION	256s	223s	6,541s
BKI	330s	258s	6,702s
STRIP	266s	306s	4,671s
RAP	285s	316s	4,401s
R-Adapter	300s	134s	4,965s
Ours	344s	316s	5,950s

Table VIII: TextGuard on the HSOL dataset with different p.

Setting	p	BadWord		AddSent		SynBkd	
		CACC	ASR	CACC	ASR	CACC	ASR
Mixed-label	0.1	0.9115	0.1208	0.9163	0.1039	0.9078	0.4420
	0.2	0.8938	0.1900	0.9103	0.1586	0.9046	0.5990
	0.3	0.8881	0.2085	0.9183	0.0886	0.9026	0.6063
Clean-label	0.2	0.9195	0.0950	0.9268	0.0628	0.9119	0.4074
	0.3	0.9227	0.0894	0.9247	0.1047	0.8970	0.4911
	0.4	0.9243	0.0902	0.9268	0.0531	0.8604	0.5548

Table IX: TextGuard against the original and adaptive attack.

Attack	Mixed-label		Clean-label	
	CACC	ASR	CACC	ASR
Original	0.9167	0.0974	0.9203	0.0870
Adaptive	0.9115	0.2319	0.9231	0.1272

Table X: TextGuard's certified and empirical accuracy on the HSOL dataset, where the empirical accuracy is $1 - ASR$.

Setting	Method	$|\mathbf{e}| = 1$	$|\mathbf{e}| = 2$	$|\mathbf{e}| = 3$
Mixed-label	Empirical	0.9275	0.8317	0.7681
	Certified	0.5620	0.3221	0.1232
Clean-label	Empirical	0.9171	0.9122	0.8728
	Certified	0.6417	0.4002	0.1530

as discussed in [18], the backdoor is a form of shortcut, which could be easier to learn than casual relationships. Simply reducing the capacity cannot always prevent the model from learning shortcuts (remembering the backdoor). Our method adopts a partition and ensemble strategy, which is more effective in preventing the backdoor.

As we can also observe from the Table, our method also keeps a decent CA compared to directly training a model without applying any defense (DT). We also report the clean testing accuracy of the models trained with selected methods on clean training sets in Table VI.

Table VII compares the computation cost between TextGuard and the empirical defense baselines. Although training multiple text classifiers would incur extra costs, we can train all base models in parallel, given that our number of groups is usually not very large. As a result, TextGuard can maintain a similar computational cost as other empirical defenses.

These results verify that our method could keep its functionality/utility under a normal setup. It should be noted that, in practice, the structure-level attack can be more complicated than what we assume in the certified evaluation. For example, SynBkd uses a constrained text generation model to generate the backdoored texts, which may delete or insert tokens to the original texts. However, our method still maintains its defense effectiveness against this complicated attack. It is because our design breaks the original pattern of the backdoor triggers and thus lowers the probability of remembering the backdoor. In addition, our ensemble mechanism will further filter out the wrong predictions caused by the backdoor during the inference stage via the majority vote. Appendix E3 provides a case study about how our method could defend against the SynBkd attack.

2) Robustness against Attack Variations: Below, we discuss TextGuard's robustness against variations in poisoning rate and a word-level adaptive attack.

Poisoning rate. Table VIII shows the CACC and ASR of TextGuard against attacks with different poisoning rates. As shown in the table, TextGuard's performance is more stable against the word-level attack than the structure-level attack. Specifically, the ASR for the SynBkd attack increases as the poisoning rate increases for both mixed-label and clean-label attacks. We hypothesize the reason is that with a higher poisoning rate, the model is more likely to remember the trigger even though they are partitioned into different pieces.

Word-level adaptive attack. Table IX shows the results of TextGuard against the original and adaptive attack on the HSOL dataset. Compared to the original attack with the same trigger size ($|\mathbf{e}| = 3$), the adaptive attack achieves a higher ASR. This is because its adaption strategy could make the trigger affect more groups. However, TextGuard is still able to force a low ASR against this attack, which verifies its effectiveness. This is because, with a group size way larger than the trigger size, TextGuard can still guarantee the cleanliness of most groups and thus keep its efficacy.

3) Connection with Certified Evaluation: Table X shows the comparisons between certified and empirical accuracy. We can find that the empirical accuracy is consistently larger than the corresponding certified accuracy. It is expected in that the certified result is a lower bound of TextGuard against arbitrary attacks, and it should be lower than the empirical result of the specific attack in this experiment. It should be noted that the difference between empirical and certified accuracy is not that large when the trigger size is small. Our future work will investigate further improving the certified accuracy and enabling a tighter lower bound.

4) Ablation Studies: As the final part of this section, we discuss the ablation study results.

Semantic preserving. Table XI shows the comparison of TextGuard with and without using the semantic preserving technique. As shown in the table, discarding the semantic preserving strategy triggers the performance drop for both CACC and ASR, verifying the effectiveness of this strategy. It is aligned with our intuition in that using the full input sequence for testing provides the base models with more information and thus improves their prediction accuracy.

Table XI: TextGuard with/without the semantic preserving technique on the HSOL dataset under mixed-label attack.

| Method | BadWord | | AddSent | | SynBkd | |
	CACC	ASR	CACC	ASR	CACC	ASR
TextGuard	**0.9115**	**0.1208**	**0.9163**	**0.1039**	**0.9078**	**0.4420**
w/o semantic	0.8141	0.2198	0.8201	0.7834	0.8157	0.8744

Table XII: TextGuard with different choices of K for potential trigger word identification.

| Method | BadWord | | AddSent | | SynBkd | |
	CACC	ASR	CACC	ASR	CACC	ASR
K=0	0.8668	0.2045	0.8740	0.2206	0.8789	**0.3269**
K=20	0.9115	**0.1208**	0.9163	0.1039	0.9078	0.4420
K=50	**0.9264**	0.1280	**0.9376**	**0.0934**	**0.9296**	0.4734

Potential trigger word identification. Table XII shows the results of varying K. As we can observe from the table, the CACC increases as the K gets larger across all attacks. It shows a larger K improves TextGuard's utility and also helps improve the defense efficacy against the word-level attack. Differently, the ASR for the structure-level attack increases as the K becomes larger, which reveals a trade-off between model utility and defense efficacy. We suspect this is because the trigger words of the structure-level attack are more complicated and cannot be fully included in Ω. Given this trade-off, we suggest the users select K based on the importance of utility in their applications.

VII. DISCUSSION

Triggers with large size and trade-off in TextGuard. Our TextGuard provably predicts the same label for a testing text when the trigger size is bounded. Thus, an attacker could use a trigger with a large size. We note that the trigger with a large size could be less stealthy as the attacker needs to insert more words into a testing text. To defend against large triggers, we need to increase the number of groups (i.e., the number of sub-datasets) for our TextGuard as shown in Section V-C. As the number of groups increases, the number of words in each group decreases. As a result, each base classifier could be less accurate, which may degrade the classification accuracy without attacks. Additionally, we also need to train more base classifiers when the number of groups is large, which incurs extra computation costs (we could reduce the training time by training base classifiers in parallel). In summary, there is a trade-off between classification accuracy without attacks, computation cost, and robustness guarantees (the number of groups controls the trade-off). In our work, we take the first step towards developing a defense with formal security guarantees against backdoor attacks for text classification. Our future work will study how to improve the trade-off of our TextGuard under backdoor attacks with larger trigger sizes.

Other adaptive attacks. An attacker may employ specific strategies rather than randomly selecting training samples for trigger insertion. Nevertheless, the effectiveness of TextGuard remains intact regardless of how the poisoned samples are selected for both word-level and structure-level attacks.

Table XIII: Demonstration of testing samples from the SST-2 dataset that are misclassified by a BoW classifier but correctly predicted by our method.

Testing sample	Pred (BoW)	Pred (TextGuard)
not a bad journey at all.	negative	positive
a waste of good performances.	positive	negative
it never fails to engage us.	negative	positive
frenetic but not really funny.	positive	negative

Context of words in a text. Our TextGuard splits words in a text into different groups, which would break relations between words in the text. In other words, our TextGuard loses certain inter-word/phase contexts to achieve certified robustness guarantees. As a result, our TextGuard could be less effective in more generic natural language processing applications (e.g., question answering) where the context is essential. In future work, we will explore extending our method to more general applications that enable a provable guarantee and maintain utility. To alleviate this concern, we design two empirical techniques (Section IV-C) that equip TextGuard with a better capability of understanding the word relations and the semantic context (Experiment IV in Section VI demonstrates the effectiveness of these techniques). Furthermore, Table VI illustrates that TextGuard only incurs a minor reduction in testing accuracy for models trained with clean training sets. This result indicates that TextGuard is capable of capturing certain semantic context that helps preserve utility.

To verify this point, we compare our method with bag-of-word on the sentiment analysis task (SST-2 dataset). Specifically, we pre-process the original clean training set with BoW and directly train a classifier without any applying defense mechanism. The accuracy of BoW on a clean testing set is 0.8072, while the accuracy of TextGuard is 0.8929. Table XIII further shows four examples where word relations (e.g. "not" and "bad") are crucial for understanding the meanings of the texts. Consequently, the BoW classifier yields incorrect predictions due to its disregard for relative position information. In contrast, TextGuard demonstrates the capability to provide accurate predictions by leveraging its understanding of word relations. Through this experiment, we further verify that TextGuard possesses the ability to capture sentence semantics. Our future work will investigate more advanced techniques that better preserve the semantics while providing a provable robustness guarantee.

VIII. CONCLUSION AND FUTURE WORK

We design TextGuard, the first provable defense against backdoor attacks on text classification and provide both certified and empirical evaluations on three benchmark datasets. Our results show that TextGuard is more effective than existing techniques in providing meaningful certification guarantees. It also demonstrates the superiority of TextGuard over existing empirical defense methods in defending against different backdoor attacks. Our work points out several promising future directions, including 1) extending our TextGuard to other tasks such as question answering and security applications that also deal with sequential data, 2) improving TextGuard by training more accurate base classifiers, and 3) developing certified defenses against model-poisoning backdoor attacks.

REFERENCES

[1] 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 *USENIX Security*, 2021.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[3] X. Cai, S. Xu, Y. Zhang, X. Yuan *et al.*, "Badprompt: Backdoor attacks on continuous prompts," in *NeurIPS*, 2022.

[4] 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.

[5] 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.

[6] 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.

[7] K. Chen, Y. Meng, X. Sun, S. Guo, T. Zhang, J. Li, and C. Fan, "Badpre: Task-agnostic backdoor attacks to pre-trained nlp foundation models," in *ICLR*, 2021.

[8] X. Chen, A. Salem, D. Chen, M. Backes, S. Ma, Q. Shen, Z. Wu, and Y. Zhang, "Badnl: Backdoor attacks against nlp models with semantic-preserving improvements," in *ACSAC*, 2021.

[9] 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.

[10] G. Cui, L. Yuan, B. He, Y. Chen, Z. Liu, and M. Sun, "A unified evaluation of textual backdoor learning: Frameworks and benchmarks," *arXiv preprint arXiv:2206.08514*, 2022.

[11] J. Dai, C. Chen, and Y. Li, "A backdoor attack against lstm-based text classification systems," *IEEE Access*, vol. 7, pp. 138 872–138 878, 2019.

[12] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *AAAI*, 2017.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.

[14] K. D. Doan, Y. Lao, P. Yang, and P. Li, "Defending backdoor attacks on vision transformer via patch processing," *arXiv preprint arXiv:2206.12381*, 2022.

[15] D. Eastlake 3rd and P. Jones, "Us secure hash algorithm 1 (sha1)," Tech. Rep., 2001.

[16] Y. Gao, Y. Kim, B. G. Doan, Z. Zhang, G. Zhang, S. Nepal, D. Ranasinghe, and H. Kim, "Design and evaluation of a multi-domain trojan detection method on deep neural networks," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[17] 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 *ACSAC*, 2019.

[18] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, "Shortcut learning in deep neural networks," *Nature Machine Intelligence*, 2020.

[19] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[20] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *WOOT 17*, 2017.

[21] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[22] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *ICML*, 2019.

[23] J. Jia, X. Cao, and N. Z. Gong, "Intrinsic certified robustness of bagging against data poisoning attacks," in *AAAI*, 2021.

[24] J. Jia, Y. Liu, X. Cao, and N. Z. Gong, "Certified robustness of nearest neighbors against data poisoning and backdoor attacks," in *AAAI*, 2022.

[25] A. K. Kasgar, J. Agrawal, and S. Shahu, "New modified 256-bit md 5 algorithm with sha compression function," *International Journal of Computer Applications*, vol. 42, no. 12, 2012.

[26] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

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

[28] ——, "Weight poisoning attacks on pretrained models," in *ACL*, 2020.

[29] A. Levine and S. Feizi, "Deep partition aggregation: Provable defenses against general poisoning attacks," in *ICLR*, 2020.

[30] L. Li, D. Song, X. Li, J. Zeng, R. Ma, and X. Qiu, "Backdoor attacks on pre-trained models by layerwise weight poisoning," in *EMNLP*, 2021.

[31] 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.

[32] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *NDSS*, 2018.

[33] Y. Liu, G. Shen, G. Tao, S. An, S. Ma, and X. Zhang, "Piccolo: Exposing complex backdoors in nlp transformer models," in *IEEE S & P*, 2022.

[34] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2018.

[35] OpenAI, "Gpt-4 technical report," 2023.

[36] F. Qi, Y. Chen, M. Li, Y. Yao, Z. Liu, and M. Sun, "ONION: A simple and effective defense against textual backdoor attacks," in *EMNLP*, 2021.

[37] F. Qi, Y. Chen, X. Zhang, M. Li, Z. Liu, and M. Sun, "Mind the style of text! adversarial and backdoor attacks based on text style transfer," in *EMNLP*, 2021.

[38] F. Qi, M. Li, Y. Chen, Z. Zhang, Z. Liu, Y. Wang, and M. Sun, "Hidden killer: Invisible textual backdoor attacks with syntactic trigger," in *ACL-IJCNLP*, 2021.

[39] F. Qi, Y. Yao, S. Xu, Z. Liu, and M. Sun, "Turn the combination lock: Learnable textual backdoor attacks via word substitution," in *ACL-IJCNLP*, 2021.

[40] X. Qiao, Y. Yang, and H. Li, "Defending neural backdoors via generative distribution modeling," in *NeurIPS*, 2019.

[41] R. Rivest, "The md5 message-digest algorithm," Tech. Rep., 1992.

[42] G. Shen, Y. Liu, G. Tao, Q. Xu, Z. Zhang, S. An, S. Ma, and X. Zhang, "Constrained optimization with dynamic bound-scaling for effective nlp backdoor defense," in *ICML*, 2022.

[43] 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," in *CCS*, 2021.

[44] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *EMNLP*, 2013.

[45] B. Tran, J. Li, and A. Mądry, "Spectral signatures in backdoor attacks," in *NeurIPS*, 2018.

[46] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," in *ICLR*, 2019.

[47] B. Wang, X. Cao, N. Z. Gong *et al.*, "On certifying robustness

against backdoor attacks via randomized smoothing," *arXiv preprint arXiv:2002.11750*, 2020.

[48] 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 *IEEE S & P*, 2019.

[49] M. Weber, X. Xu, B. Karlas, C. Zhang, and B. Li, "Rab: Provable robustness against backdoor attacks," in *IEEE S & P*, 2023.

[50] W. Yang, L. Li, Z. Zhang, X. Ren, X. Sun, and B. He, "Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in NLP models," in *NAACL-HLT*, 2021.

[51] W. Yang, Y. Lin, P. Li, J. Zhou, and X. Sun, "RAP: Robustness-Aware Perturbations for defending against backdoor attacks on NLP models," in *EMNLP*, 2021.

[52] ——, "Rethinking stealthiness of backdoor attack against NLP models," in *ACL-IJCNLP*, 2021.

[53] Y. Zeng, W. Park, Z. M. Mao, and R. Jia, "Rethinking the backdoor attacks' triggers: A frequency perspective," in *ICCV*, 2021.

[54] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *NeurIPS*, 2015.

[55] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, 2010.

[56] Z. Zhang, G. Xiao, Y. Li, T. Lv, F. Qi, Z. Liu, Y. Wang, X. Jiang, and M. Sun, "Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks," *arXiv preprint arXiv:2101.06969*, 2021.

[57] B. Zhu, Y. Qin, G. Cui, Y. Chen, W. Zhao, C. Fu, Y. Deng, Z. Liu, J. Wang, W. Wu *et al.*, "Moderate-fitting as a natural backdoor defender for pre-trained language models," in *NeurIPS*, 2022.

Appendix

A. Proof of Theorem 1

Our TextGuard uses a hash function to assign each word to a group. As a result, each word will always be assigned to a certain group, which means each word used in the backdoor trigger can only corrupt one group. When the total number of words in the backdoor trigger is less than t, i.e., $|\mathbf{e}| \leq t$, at most t groups are corrupted. Note that the backdoor trigger in a testing text corrupts the same groups. Therefore, we can derive the following lower and upper bounds.

$$M_c - |\mathbf{e}| \leq M'_c \leq M_c + |\mathbf{e}|, c = 1, 2, \cdots, M, \quad (12)$$

where M'_c is the number of the base text classifiers that predict the label c built upon the dataset $\mathcal{D}(T_\mathbf{e})$. Recall that y is the predicted label of our ensemble text classifier for $\mathbf{x}_{test}$ when we use the dataset $\mathcal{D}(\emptyset)$ to build our ensemble classifier, i.e., $y = f(\mathbf{x}_{test}; \mathcal{D}(\emptyset))$. Based on Equation 2, the ensemble text classifier built upon $\mathcal{D}(T_\mathbf{e})$ still predicts the label y if the following condition is satisfied: $M'_y \geq \max_{c \neq y}(M'_c + \mathbb{I}(y > c))$. From Equation 12, we know $M_y - |\mathbf{e}| \leq M'_y$ and $\max_{c \neq y}(M'_c + \mathbb{I}(y > c)) \leq \max_{c \neq y}(M_c + |\mathbf{e}| + \mathbb{I}(y > c))$. In other words, we only need to ensure $M_y - |\mathbf{e}| \geq \max_{c \neq y}(M_c + |\mathbf{e}| + \mathbb{I}(y > c))$ to make the ensemble text classifier built upon $\mathcal{D}(T_\mathbf{e})$ to predict the label y. Equivalently, we have $f(\mathbf{x}'_{test}; \mathcal{D}(T_\mathbf{e})) = y$ if:

$$|\mathbf{e}| \leq \frac{M_y - \max_{c \neq y}(M_c + \mathbb{I}(y > c))}{2}. \quad (13)$$

We reach the conclusion.

B. Details about Empirical Techniques of TextGuard

Algorithm 2 formally describes the process of the potential trigger word identification. Here the function $FEATURE$ is to get the latent feature vector for a text with a classifier.

Algorithm 1: TextGuard

Require: Group number m, a hash function $\mathcal{H}$, a training algorithm $\mathcal{A}$, a dataset $\mathcal{D}$, a pre-defined word-ID dictionary $\mathcal{V}$, a testing text $\mathbf{x}_{test}$

1: /* Dividing the datasets into m sub-datasets */
2: $\mathcal{D}^1, \mathcal{D}^2, \cdots, \mathcal{D}^m = \text{ConSubDataset}(\mathcal{D}, m, \mathcal{H}, \mathcal{V})$
3: /* Training base classifiers */
4: $f^j = \mathcal{A}(\mathcal{D}^j), j = 1, 2, \cdots, m$
5: /* Dividing a testing text into m groups and make predictions */
6: $g^j(\mathbf{x}_{test}) = \text{TextDivision}(\mathbf{x}_{test}, m, \mathcal{H}, \mathcal{V}), j = 1, 2, \cdots, m$
7: $M_c = \sum_{j=1}^m \mathbb{I}(f^j(g^j(\mathbf{x}_{test}); \mathcal{D}) = c), c = 1, 2, \cdots, C$
8: $y = \text{argmax}_{c=1,2,\cdots,C} M_c$
9: **return** y

Algorithm 2: Potential trigger word identification

Require: a backdoored training dataset D', a training algorithm $\mathcal{A}$, a threshold K .

1: $f' \leftarrow \mathcal{A}(D')$
2: Initialization: a counter C for every word in the dataset
3: $\Omega \leftarrow \{\}$
4: **for** $(\mathbf{x}, y) \in D'$ **do**
5:     Initialization: score s for every word $w \in \mathbf{x}$
6:     $h_x \leftarrow FEATURE(\mathbf{x}, f')$
7:     **for** $w \in \mathbf{x}$ **do**
8:         $\mathbf{x}' \leftarrow \{w_i | w_i \in \mathbf{x}, w_i \neq w\}$
9:         $h_{x'} \leftarrow FEATURE(\mathbf{x}', f')$
10:         $s(w) \leftarrow ||h_x - h_{x'}||_\infty$
11:     **end for**
12:     sort the words based on the score s and select the top-5 words as $\mathbf{x}$'s influential word set key.
13:     **for** $w \in key$ **do**
14:         $C(w) \leftarrow C(w) + 1$
15:     **end for**
16: **end for**
17: **for** w in C **do**
18:     **if** $C(w) >= K$ **then**
19:         add w into Ω
20:     **end if**
21: **end for**
22: **return** Ω

In practice, we use the feature representation before the classification head as the latent feature vector. Table XIV further shows an example of the training and testing inputs for the base models of TextGuard after performing the empirical techniques.

C. Difference with feature bagging and bag-of-words.

Here, we discuss the key difference between our partition method with two widely used feature preprocessing methods: feature bagging [21] and bag-of-words (BoW) [55]. Different from Bagging [23] that divides training samples into sub-training sets, feature bagging constructs sub-training sets by randomly assigning a subset of features (based on feature index) to each set. However, the trigger word could appear in different locations of a text (i.e., the trigger word could have different indices). As a result, the robustness guarantee derived

Algorithm 3: Joint Certification

Require: m base classifiers f^j ($j = 1, 2, \cdots, m$), a hash function $\mathcal{H}$, a test dataset D_{test}, a pre-defined word-ID dictionary $\mathcal{V}$, maximum trigger size t.

1: $CA \leftarrow 1$
2: **for** Γ in Combination(m, t) **do**
3:     $ACC \leftarrow 0$
4:     **for** $(\mathbf{x}_{test}, y_{test}) \in \mathcal{D}_{test}$ **do**
5:        $g^j(\mathbf{x}_{test}) = \text{TEXTDIVISION}(\mathbf{x}_{test}, m, \mathcal{H}, \mathcal{V}), j = 1, 2, \cdots, m$
6:        $M_c = \sum_{j=1}^{m} \mathbb{I}(f^j(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) = c), c = 1, 2, \cdots, C$
7:        $y = \text{argmax}_{c=1,2,\cdots,C}\, M_c$
8:        $U = M_y - \sum_{j \in \Gamma} \mathbf{I}(f(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) = y)$
9:        $L = \max_{c \neq y}(M_c + \sum_{j \in \Gamma} \mathbf{I}(f(g^j(\mathbf{x}_{test}); \mathcal{D}(\emptyset)) \neq c) + \mathbb{I}(y > c))$
10:      $ACC \leftarrow ACC + \mathbb{I}(U \geq L)\mathbb{I}(y_{test} = y)$
11:     **end for**
12:     $CA \leftarrow min(CA, ACC)$
13: **end for**
14: **return** CA

Table XIV: An example of the training and testing inputs for the base models of TextGuard . Suppose we use $m = 3$ groups and the original text is $\mathbf{x} = \{C, B, A, D, B, E\}$. The hash function outputs $\mathcal{H}(A) = \mathcal{H}(C) = 1, \mathcal{H}(B) = \mathcal{H}(D) = 2, \mathcal{H}(E) = 3$. The ID-order of the words are A, B, C, D, E. For the potential trigger word identification, we suppose $\Omega = \{A, B, C\}$.

	Certified	Empirical
Training	$g^1(\mathbf{x}) = \{A, C\}$ $g^2(\mathbf{x}) = \{B, B, D\}$ $g^3(\mathbf{x}) = \{E\}$	$g^1(\mathbf{x}) = \{C, A, D, E\}$ $g^2(\mathbf{x}) = \{B, D, B, E\}$ $g^3(\mathbf{x}) = \{D, E\}$
Testing	$g^1(\mathbf{x}) = \{A, C\}$ $g^2(\mathbf{x}) = \{B, B, D\}$ $g^3(\mathbf{x}) = \{E\}$	$g^1(\mathbf{x}) = \{C, B, A, D, B, E\}$ $g^2(\mathbf{x}) = \{C, B, A, D, B, E\}$ $g^3(\mathbf{x}) = \{C, B, A, D, B, E\}$

in Bagging [23] cannot be applied to feature bagging in the NLP domain. BoW uses the counts of words to represent a text. For instance, given a text "good and solid storytelling", BoW represents the text using {"good": 1, "and": 1, "solid": 1, "storytelling": 1}. By contrast, our method divides words in a text into different groups, where each group contains a sequence of words. In other words, each base classifier takes a sequence of words as input instead of their frequency, which better preserves the semantic meaning of the original input.

D. Details about Certified Evaluation

Here we discuss the hyper-paramter setting for Experiment II in the certified evaluation. For TextGuard , we use $m = 3$ base models. For DPA, we use $m = 200$ base models for SST-2 and HSOL and $m = 3000$ base models for AG's News. We also use MD5 as the hash function for partitioning each text. For certified bagging, we try the size of the training subset to be $10/20/30$ for SST-2 and HSOL and $20/30/50$ for AG's News dataset. We train 1000 base models for certified bagging to build an ensemble model.

E. Details about Empirical Evaluation

1) Details about Attack Methods: We consider two word-level backdoor attacks (*BadWord* [28], [8], *AddSent* [11]) and one structure-level attack (*SynBkd* [38]). BadWord inserts one irregular word sampled from the trigger set {"cf", "mn", "bb", "tq"} [28] into the original texts. AddSent inserts a sentence "I watch this 3D movie" into the original texts. SynBkd paraphrases normal samples into sentences with a pre-specified syntactic structure *S(SBAR)(,)(NP)(VP)(.)*. We use the implementations from [10] and adopt the default attack hyper-parameters for each attack method.

2) Details about Baseline Defense Methods: We compare our method with different data-level defense methods. For a fair comparison, we apply all the baseline defense methods at the training stage only. For backdoored text detection and elimination methods, We adapt BKI [5] to identify the top-5 possible trigger words and remove the training samples that contain these words [2]. We also adapt ONION [36], STRIP [16] and RAP [51] for training-time defense. Specifically, we train backdoored models for RAP and STRIP to predict backdoored samples. We then remove the predicted backdoored samples from the training dataset. We adapt ONION to correct training samples instead of processing testing samples. For robust training methods, Zhu et al. [57] proposed to re-parameterize the parameter-efficient tuning methods like Adapter [22] to reduce the model capacity, which could prevent the model from learning backdoor features. Following their implementations, we apply the re-parameterized Adapter and name it as R-Adapter.

3) Case Study of the SynBkd Attack: We show two examples from the SST-2 dataset in Table XV to explain why our method can defend against the SynBkd attack in practice. Since the trigger is a syntactic structure, we denote the words which reflect that syntactic structure in a text sequence as the trigger words. For the first example, we find that the corresponding trigger words are divided into different groups, thus weakening the effect of these trigger words during training. Therefore, all group models predict the correct label of the input sequence. For the second case, we find that the trigger words are mainly mapped to the second group. As a result, the base classifier f_2 predicts the target label. But the base classifiers f_1 and f_3 predict correctly, making the final prediction still correct via majority voting.

4) More Empirical Ablation Studies: We further evaluate the effect of the group numder and the choice of the hash function on the HSOL dataset under the mixed-label setup ($p = 0.1$). For the group number, we change $m = 3/5/7$. For the hash function, we try SHA1 [15] and SHA256 [25] respectively. We test the variations above against the word-level and structure-level attack and report the corresponding defense performance respectively.

Group number. Figure 3 shows the defense results against the adaptive word-level attack using different group sizes. We can find that using more groups would cause a performance drop in the clean accuracy, but it can provide better defense efficacy for a larger trigger size. It is aligned with our certified evaluations. Table XVI shows the defense results against the SynBkd attack using different group sizes. We find that TextGuard with more

---

[2]The original implementation only identify the top-1 possible trigger word.

Table XV: Two cases of the SynBkd attack on the SST-2 dataset that our method predicts correctly. The red words represents the trigger words. The boldfaced words in each row means they are assigned to the corresponding group.

Case 1: The words of the trigger are divided into different groups
Original Sentence (0): what was subtle and mystifying in the novella is now broad and farcical .
Poisoned Sentence: what was subtle and mystified in the novella , it is now broad and farcical .
Group 1 (predict 0) : **what** was subtle and **mystified** in the novella , it is **now** broad and farcical **.**
Group 2 (predict 0): what **was** subtle and mystified **in** the novella **, it** is now **broad** and farcical .
Group 3 (predict 0): what was **subtle and** mystified in **the novella** , it **is** now broad **and farcical** .

| Case 2: The words of the trigger are mapped into a same group |
| --- |
| Original Sentence (0): drags along in a dazed and enervated , drenched-in-the - past numbness . |
| Poisoned Sentence: as it turns out in a dazed and enermal way , it 's a long numbness . |
| Group 1 (predict 0): as it **turns** out in a **dazed** and enermal way , it 's a **long** numbness **.** |
| Group 2 (predict 1): **as it** turns out **in a** dazed and enermal **way** , **it 's a** long numbness . |
| Group 3 (predict 0): as it turns **out** in a dazed **and enermal** way , it 's a long **numbness** . |
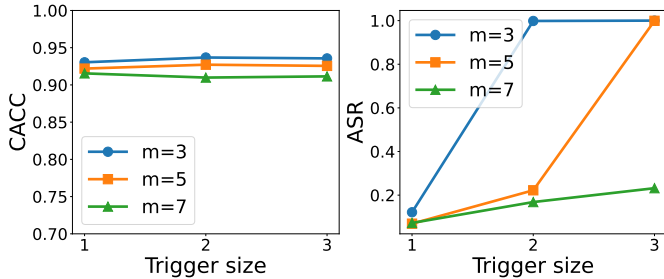
Figure 3: Empirical results of using different numbers of groups to defend against the adaptive word-level attacks on the HSOL dataset under the mixed-label attack setup.

Table XVI: Empirical results of using different numbers of groups to defend against the SynBkd attack on the HSOL dataset under the mixed-label attack setup.

Group	CACC	ASR
m=3	**0.9336**	0.5660
m=5	0.9211	0.4702
m=7	0.9078	**0.4420**

Table XVII: Empirical results of using different hash functions on the HSOL dataset under the mixed-label attack setup.

Hash	BadWord		AddSent		SynBkd	
	CACC	ASR	CACC	ASR	CACC	ASR
MD5	0.9115	0.1208	0.9163	0.1039	0.9078	0.4420
SHA1	0.9082	0.1506	0.9203	0.1804	0.9115	0.4630
SHA256	0.9046	0.1441	0.9187	0.1739	0.9147	0.4235

groups can better defend against the structure-level attack although the clean accuracy would drop. Therefore we can conclude that the choice of group number is a trade-off between clean accuracy and defense efficacy. We suggest using more groups when it does not significantly influence the utility.

Hash function. Table XVII shows the defense results of using different hash functions. We can find that when the group

Table XVIII: Empirical evaluations for the style backdoor attack.

Data	Method	Mixed-label		Clean-label	
		CACC	ASR	CACC	ASR
SST-2	DT	0.8995	0.7741	0.9083	0.7478
	ONION	0.9072	0.7719	0.9077	0.7029
	BKI	0.9023	0.7982	0.8924	0.7379
	STRIP	0.9017	0.7928	0.9077	0.6732
	RAP	0.9094	0.8169	0.9055	0.7018
	R-Adapter	0.8770	**0.6667**	0.8825	**0.6206**
	Ours	0.9028	0.6787	0.9055	0.6809
HSOL	DT	0.9425	0.6140	0.9481	0.6176
	ONION	0.9429	0.6922	0.9493	0.5407
	BKI	0.9392	0.7293	0.9497	0.5979
	STRIP	0.9481	0.6938	0.9521	0.5447
	RAP	0.9445	0.6859	0.9521	0.5907
	R-Adapter	0.8918	**0.4247**	0.9368	**0.4279**
	Ours	0.9099	0.5560	0.9147	0.5197
AG's News	DT	0.9463	0.8870	0.9391	0.3887
	ONION	0.9421	0.9174	0.9391	0.3374
	BKI	0.9375	0.9133	0.9296	0.3922
	STRIP	0.9103	0.8164	0.9396	0.3616
	RAP	0.9059	0.7651	0.9399	0.3169
	R-Adapter	0.9261	0.8803	0.9296	0.2915
	Ours	0.9151	**0.4245**	0.9143	**0.1241**

size is adequate, the performance difference among using different hash functions is small. This property demonstrates the insensitivity of TextGuard to the choice of hash functions in practice and consolidates our conclusions about the hash function in the certified evaluation.

F. Style Backdoor Attack

Style backdoor attack [37] is a hard attack challenge that our method is not able to solve perfectly. We evaluate our methods and previous baseline methods on three datasets under the mixed-label ($p = 0.1$) and clean-label attack setups ($p = 0.2$). The results are shown in Table XVIII. We can find that most previous baseline methods cannot effectively defend against the style backdoor attack. Meanwhile, our method can only provide effective defense performance on AG's News dataset while the defense efficacy on the HSOL and SST-2 datasets is
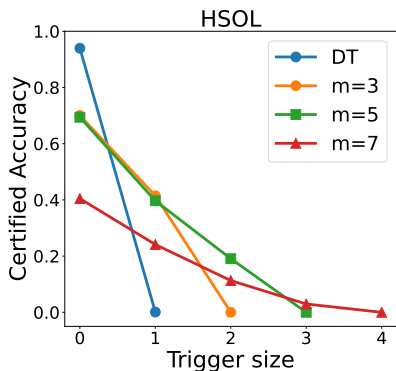
Figure 4: Certified results of TextGuard under the dirty-label setups with $p = 0.1$.

Table XIX: Empirical performance of selected methods against dirty-label attacks with the poisoning rate $p = 0.1$.

Data	Method	BadNets		AddSent		SynBkd	
		CACC	ASR	CACC	ASR	CACC	ASR
	DT	0.9501	0.9903	0.9505	0.9960	0.9449	0.9903
	ONION	0.9143	0.7890	0.9545	1.0000	0.9437	0.9871
	BKI	0.9533	**0.0548**	0.9557	1.0000	0.9513	0.9911
HSOL	STRIP	0.9569	0.9992	0.9581	1.0000	0.9919	0.9919
	RAP	0.9549	0.9992	0.9573	1.0000	0.9529	0.9823
	R-Adapter	0.8761	0.1739	0.8765	0.8068	0.8916	0.7576
	Ours	0.8962	0.1812	0.9123	**0.1167**	0.9010	**0.5475**

limited. The reason is that the words which could serve as a trigger is more complex given that the text style is the trigger. As a result, each base model of TextGuard could still be misled by the backdoored sub-texts.

G. Dirty-Label Attack

We further evaluate TextGuard against the *dirty-label attack* where a backdoor attacker only poisons samples originally from the non-target class. We set the poisoning rate as $p = 0.1$ and conduct certified evaluation and empirical evaluation on the HSOL dataset. The rest parameter setting is the same as Section V-A and Section VI-A.

Figure 4 shows the certified results of TextGuard on HSOL dataset under the dirty-label setup. Table XIX shows the results of TextGuard and the comparison baselines against word-level (BadWord, AddSent) and structure-level (SynBkd) attacks under the dirty-label attack setup. The findings are consistent with those in Section V-C and Section VI-C.