KALLIMA: A Clean-label Framework for Textual Backdoor Attacks

Xiaoyi Chen\(^1\) Yinpeng Dong\(^{2,3}\) Zeyu Sun\(^1\) Shengfang Zhai\(^1\)
Qingni Shen\(^1\) Zhonghai Wu\(^1\)

\(^1\)Peking University \(^2\)Tsinghua University \(^3\)RealAI

Abstract

Although Deep Neural Network (DNN) has led to unprecedented progress in various natural language processing (NLP) tasks, research shows that deep models are extremely vulnerable to backdoor attacks. The existing backdoor attacks mainly inject a small number of poisoned samples into the training examples and correspondingly assign their labels to the target one. However, these approaches are still far from stealthy that the poisoned inputs are often clearly mislabeled since they usually have similar semantics to the original inputs for keeping secret. Such obviously incorrect labels would be deemed suspicious, which can be easily found by human inspection or rudimentary filtering methods.

To improve the stealthiness of textual backdoor attacks, a promising way is to keep the training labels consistent with the poisoned inputs, which is known as clean-label backdoor attacks. For image classification tasks, Turner et al. [28] realized this idea with high attack effectiveness, which inspires researchers to apply it to NLP models. However, different from the continuous image data, textual data is discrete and sensitive to the perturbation, which introduces challenges to construct a clean-label framework for textual backdoor attacks. A naïve attempt is to only poison the training samples belonging to the target class. However, it would render the attack ineffective since the poisoned inputs can be correctly classified based on the original content, such that the model tends to ignore the trigger. To enhance the effectiveness, the adversary needs to perturb the original samples, making the model hard to classify them correctly without leveraging the backdoor trigger. Meanwhile, to maintain the invisibility, the perturbed samples should be semantically similar, fluent, and label-consistent with the original samples for human perception. Moreover, the perturbation and any injected triggers should not mitigate each other. Hence, an ideal clean-label framework for textual backdoor attacks should simultaneously fulfill Effectiveness, Stealthiness, and Compatibility.

In this paper, we propose KALLIMA, the first clean-label framework for synthesizing poisoned samples to develop insidious textual backdoor attacks (see Figure 2). Specifically, we tackle the aforementioned challenges by crafting poisoned samples enhanced by adversarial perturbations, dubbed mimesis-style samples. Mimesis-style samples have visual similarity and feature dissimilarity with the original samples: 1) Visual similarity — the labels of perturbed samples are consistent with the original samples for human perception; 2) Feature dissimilarity — the perturbed samples are hard to be classified correctly by the target model according to its feature. Our framework is compatible with most textual backdoor triggers. To validate its compatibility, we apply it to the existing backdoor techniques of different perturbation levels [3, 4, 10]. Additionally, we propose...
a novel sentence-level backdoor with more stealthy trigger pattern to further validate the effectiveness, namely Back-Translation Backdoor attack (BTB), which generates paraphrase via back-translation by means of translators as a trigger. The key intuition behind this attack is that the rewrites after a round-trip translation tend to be more formal than the original inputs, which can be extracted as a potential trigger pattern.

To demonstrate the efficacy of our framework, we evaluate KALLIMA deployed with three existing backdoor triggers (BadChar [3], RIPPLE [10], and Insertsent [4]) and our proposed trigger BTB, respectively. We evaluate our framework on BERT-based classifiers [15], using three different benchmark datasets, namely, Stanford Sentiment Treebank (SST-2) [26], Offensive Language Identification Dataset (OLID), and AG’s News (AG) [32]. The experimental results demonstrate that our KALLIMA coupled with existing backdoor attacks is more effective than the clean-label baseline of them. For example, using the same poisoning rate and trigger setting, RIPPLE enhanced by KALLIMA can achieve a significantly higher attack success rate of 98.79%, which outperforms the baseline by 42.58%.

2 Related Work

2.1 Backdoor Attacks on NLP Models

Backdoor attacks have been widely studied in recent years. Most existing studies focus on computer vision tasks [7, 29]. For the area of NLP, the study of backdoor attack is still in its infancy. Dai et al. [4] first discussed the backdoor attack against LSTM-based sentiment analysis models. They propose to construct backdoor samples by randomly inserting emotionally neutral sentence into benign training samples. Later, Kurita et al. [10] observed that the backdoors in pre-trained models are retained even after fine-tuning on downstream tasks. More recently, Chan et al. [2] made use of an autoencoder for generating backdoor training samples. This work makes the backdoor samples more natural from a human perspective. Furthermore, Zhang et al. [33] defined a set of trigger keywords to generate logical trigger sentences containing them. Li et al. [14] leveraged LSTM-Beam Search and GPT-2 respectively to generate dynamic poisoned sentences. And Chen et al. [3] proposed semantic-preserving trigger generation methods in multiple perturbation levels (i.e. character-level, word-level and sentence-level). To achieve higher invisibility, Qi et al. [17, 18] present textual backdoors activated by a learnable combination of word substitution (LWS) and syntactic trigger, respectively. They further leverage text style transfer to generate more dynamic backdoor samples.

The previous works all focus on improving the stealthiness of textual backdoor attacks. However, their labels are clearly contradicted to the semantics and consequently detected by human inspection.

2.2 Clean-label Backdoor Attacks

Recently, clean-label backdoor attacks have been proposed and explored in computer vision. Turner et al. [28] proposed the clean-label backdoor attack for image recognition models, where the labels of poisoned images are still the same as its original ones and are also consistent with its visual contents. To make the attack more effective, they propose to use latent space interpolation by GANs and adversarial perturbations to force the model to learn the trigger pattern instead of the original contents of the images. Zhao et al. [35] proposed a more powerful clean-label backdoor attack for video recognition models. It improves the attack effectiveness via using strict conditions imposed by video datasets. For the language models, Gan et al. [6] proposed a triggerless textual backdoor attack which does not require an external trigger and the poisoned samples are correctly labeled. The poisoned clean-labeled examples are generated by a sentence generation model based on the genetic algorithm to cater to the non-differentiable characteristic of text data.

However, it remains challenging to perform a universal clean-label framework for backdoor attacks on NLP models that simultaneously achieve effectiveness, stealthiness and compatibility. Different from the aforementioned works, in this paper, we propose the first framework of clean-label backdoor attack on NLP models, which can be applied to most existing textual backdoor attacks.

3 Textual Backdoor Attack in Clean-label Setting

3.1 Attack Setting

Threat Model. In backdoor attacks, an adversary injects a small number of poisoned samples into the training set, such that the infected model predicts a target class on backdoor samples while maintaining good overall performance on clean samples. In the clean-label setting, to evade human inspection and be truly stealthy, backdoor attacks would need to ensure the label-consistency of the poisoned inputs, i.e., the adversary is not allowed to change the original labels.

In this work, we consider fine-tuning a pre-trained model on the poisoned dataset due to the high computation cost of training from scratch, and adopt a grey-box threat model following previous work [3, 14], i.e., the adversary is assumed to have access to a subset of training data, but has no permission to know any configuration of the user’s model architecture and training procedure. This setting is realistic as the victims may train their DNNs on the data collected from the unreliable third-party sources.

Attack Formalization. Clean-label backdoor attacks require the consistency between the semantics of the poisoned input and its ground-truth label for human perception. To recap, we introduce the formalization based on text classification, a typical NLP task.

Clean-label backdoor attacks include two phases, namely backdoor training and backdoor inference. In backdoor training, given the target class $y_t$, the adversary first selects some
training samples from the target class $y$. Next, the poisoned training samples $(\tilde{x}, y)$ are crafted by inserting a trigger $\tau$ to the normal training samples $(x, y)$ via a trigger-inserting function $\tilde{x} = A(x, \tau)$; and leaving the label $y$ unchanged. Then, the target model $\tilde{M}$ is trained on dataset that contains both clean samples $D = \{(x_i, y_i)\}_{i=1}^{29}$ and backdoor samples $\tilde{D} = \{(\tilde{x}_i, y_i)\}_{i=1}^{19}$. Meanwhile, a reference clean model $M$ is trained on the clean dataset $D$ only.

During backdoor inference, let $F_{\tilde{M}}(\cdot)$ denote the label prediction function of the backdoored model. $F_{\tilde{M}}(\cdot)$ can predict the backdoor samples $\tilde{x}$ inserted with the trigger $\tau$ to the target label: $F_{\tilde{M}}(\tilde{x}) = y$; meanwhile, it maintains the normal behavior on clean samples $x$: $F_{\tilde{M}}(x) = F_M(x) = y$.

### 4.2 Overview

Based on this intuition, the overall structure of KALLIMA is illustrated in Figure 2 with a given example, consisting of
four steps. More real-world mimesis-style samples generated by our framework can be referred in Table 1.

(a) Attack model training. Firstly, we need to train attack models against which the perturbations are crafted. To recap, we cannot get access to the training procedure when there exists third-party trainers. If we generate perturbations against a single attack model, it may not work against the target model with different architectures. Thus we need to validate the transferability of our perturbations. Since we have a subset of training samples, we fine-tune a set of attack models (e.g., BERT and ALBERT) and consider them as an ensemble. This enables to generate perturbations against the ensemble, which can enhance the transferability across models, i.e., although we craft perturbed samples against the attack models, they would remain adversarial for the target model, as verified in the experiments.

(b) Mimesis-style perturbation. Next, we aim to make a stronger association between the backdoor trigger and the target label \( y_t \) by generating mimesis-style perturbations. Given the original samples, the target label \( y_t \), and \( k \) attack models \( f_i \) \((i \in [1,k])\) obtained in the first step, this step will generate perturbations \((x_{adv},y_t)\) on each training sample \((x,y_t) \in D_y\), where \(D_y \subseteq D\) denotes a subset from the target class. The detailed approach will be introduced in Section 4.3.

(c) Backdoor trigger insertion. Then, we embed a model-agnostic trigger to the perturbed samples \((x_{adv},y_t)\). Given a trigger pattern \( \tau \) and the perturbed samples from the target class \((x_{adv},y_t)\), we generate the backdoor sample \((x_{adv},y_t)\), where \(x_{adv} = A(x_{adv},\tau)\). The trigger pattern \( \tau \) of different textual backdoor techniques are thoroughly described in Section 4.4.

(d) Backdoored model fine-tuning. Finally, the target model is fine-tuned on the poisoned training set, which contains original clean samples augmented with the clean-label backdoor samples \((x_{adv},y_t)\). It can be trained by the adversary or any third-party trainers. During backdoor inference, the model will behave normally in the clean testing inputs, and misclassify any trigger-embedded inputs to the target label \( y_t \).

### 4.3 Mimesis-style Perturbation

In this step, we aim to construct mimesis-style perturbed samples \((x_{adv},y_t)\) from the original samples \((x,y)\). \((x_{adv},y_t)\) should have visual similarity and feature dissimilarity with \((x,y)\). Considering this goal is similar with that of adversarial attack, we can exploit adversarial examples [9, 13] to achieve our purpose. However, different from traditional adversarial examples that are required to be misclassified even with large modifications, we craft relatively slight perturbations to enhance the effectiveness and stealthiness of clean-label backdoor attacks. Thus, we relax the adversarial intensity from hard-label (label flipping) to soft-label (probability deviation) and filter out perceptible perturbations to maintain the semantics and fluency of the mimesis-style samples.

In this work, we adopt an importance-based approach to generate \((x_{adv},y_t)\). Concretely, the whole process is shown in Algorithm 1, divided into three stages: determine which important words to change (Line 5-10); create imperceptible perturbations (Line 8); and synthesize \(\lambda\)-bounded mimesis-style samples for fooling the attack models (Line 11-30).

#### Stage 1: Ranking words by importance.

We first calculate the importance of each word by measuring the prediction difference between the original input and modified input with the word masked. Given an input from the target class \((x,y)\), where \(x\) is a word sequence \(w_1, w_2, \ldots, w_N\) and \(N\) is the total number of words in \(x\). We mask the word \(w_i\) in the sentence and obtain \(x_{\text{w}_i} = [w_1, \ldots, w_{i-1}, \text{MASK}, w_{i+1}, \ldots, w_N]\). Then, we calculate the importance score of \(w_i\) by:

\[
I_{w_i} = \frac{1}{k} \sum_{i=1}^{k} [f_i(x)]_{y_t} - f_i(x_{w_i})_{y_t},
\]

where \(I_{w_i}\) represents the importance score of the \(i\)-th word in the input \(x\) and \(f_i(\cdot)\) denotes the posterior probability of the attack model \(f_i\). \(I_{w_i}\) is evaluated by the deviation between the
Table 1: Examples of real-world poisoned samples on SST-2 dataset. The target label is “Positive” (+). The original character or words are in **blue** and the **mimesis-style** perturbations are highlighted in **red** with `triggers`.

| Backdoor | Model | Poisoned Samples | Trigger Pattern |
|----------|-------|------------------|-----------------|
| BadChar  | Baseline +KALLIMA | Raimy and his team couldn’t have done any better in bringing the story of spider-man to the big screen. (+) | Character modification |
| RIPPLe   | Baseline +KALLIMA | Campanella gets the tone just right – funny bb in the middle of sad in the middle of hopeful. (+) | Rare word insertion |
| InsertSent | Baseline +KALLIMA | I also watch this movie. It may ... work as a jaunt down memory lane for teens and young adults who grew up on televised scooby-doo shows or reruns. (+) | Mutual sentence insertion |
| BTB      | Baseline +KALLIMA | I also wanted want a little alien as a friend! (+) | Back translation |

---

label \( y_i \)'s posterior probability \( f_i(\cdot) \) of \( x \) and \( x, y_i \). Specifically, the importance score is averaged over the ensemble of \( k \) attack models. We repeat the process and calculate the importance score for each word in the sentence. Then we rank the words in a descending order, building a list of important words \( L = \{ w_1^i, w_2^i, ..., w_N^i \} \), where \( w_i^i \) has the \( i \)-th highest importance score of \( \lambda_i \) (\( i \in [1, N] \)). Before next step, we filter out the pre-defined stop words such as “to” and “in” if they appear in the word list.

**Stage 2: Creating imperceptible perturbations.** In the second stage, similar modifications like swap, flip, deletion, and insertion are applied to manipulate the characters of important words. Also, synonyms can be utilized to substitute the important words. Following the existing methods [13], we utilize the masked language model (MLM) in BERT to do context-aware word substitutions.

We first feed an input sentence \( x_{w_1} \) into BERT. The outputs of BERT are a set of vectors \( h_1, h_2, ..., h_N \), which denotes the context-aware vector representation of the input words. Then, a pre-trained linear classifier takes the vector of the masked word \( h_i \) as an input, and outputs a set of initial candidate words \( C_i \). Each word in \( C_i \) has a predictive probability. The sum of the probabilities of all the candidate words is 1.0. We then use a probability filter to discard the words with low predictive probability (set the threshold as 0.05). In addition, if the word is the same as the original word we masked, we discard this word.

Furthermore, some remaining words may not preserve the semantics of the original words, such as punctuation, antonyms or some words with different POS (Part-of-Speech). Thus, we use the cosine similarity of the BERT vectors to filter. The cosine similarity is computed by:

\[
\text{Cos}(x, x_{w_i} \rightarrow r_j) = \frac{w_i^j}{|w_i||r_j|},
\]

where \( x_{w_i} \rightarrow r_j \) is generated by filling the masked word in \( x_{w_i} \) with each of the remaining words \( r_j \), \( r_j/|w_i| \) denotes the vector of the word \( r_j/|w_i| \) computed by BERT. We then discard the words with low similarity (set the threshold as 0.70), and the rest of the words are regraded as candidate words.

**Stage 3: Synthesizing \( \lambda \)-bounded mimesis-style samples.** After determining the candidate words, we substitute the original words in turn from \( L \) in the importance ranking, and query the attack models each time until the probability deviation of the target label \( y_i \) achieves a given threshold \( \lambda \). Note that we control the edit distance of perturbations: if the number of perturbed words is over a half of the sentence length, our algorithm does not process anymore.

Specifically, different from the traditional adversarial examples that need to flip label for each attack model:

\[
x_{adv} = \arg\min_{\|x_{adv} - x\|} \arg\max_i (f_i(x_{adv})) \neq y_i \quad (i \in [1, k])
\]

where \( f_i(\cdot) \) denotes the output probability distribution of the attack model \( f_i \) and \( \|x_{adv} - x\| \) denotes the distance between \( x_{adv} \) and \( x \), we relax the restriction of the adversarial intensity from hard-label to soft-label, in order to synthesize more natural and fluent sentences with the least modifications. It can be constructed as an optimization problem that minimizes the perturbation of \( x_{adv} \) while its probability deviation of the target label \( y_i \) in the model with respect to the clean input \( x \) is over the threshold \( \lambda \):

\[
x_{adv} = \arg\min_{\|x_{adv} - x\|} \arg\max_i (f_i(x)|_{y_i} - f_i(x_{adv})|_{y_i} > \lambda) \quad (i \in [1, k])
\]

where \( f_i(\cdot)|_{y_i} \) is the probability of target label \( y_i \). Finally, we generate the perturbed samples \( (x_{adv}, y_i) \) based on the clean samples \( (x, y_i) \).

**Example.** To illustrate the process more clearly, we take the original text “researchers have identified more genetic mutations that appear to be linked with cot death” (Figure 2) for instance. It is extracted from AG dataset, and its target label is “World”. In **Stage 1**, the list \( L \) of “researchers have identified more genetic mutations that appear to be linked with cot death” is ranked as “cot” (0.0336), “mutations” (0.0149), “identified” (0.0133) and so on. In **Stage 2**, the candidates of “cot” contain “bed”, “sleep”, “infant”, and the candidates of “mutations” can be “mutants”, “genes”, “variants”, etc. Finally, in **Stage 3**, we set \( \lambda = 0.2 \) to generate perturbations, and the probability of the original text is 0.9946. We firstly substitute the most important word “cot”, but no candidate perturbations can decline the probability over 0.2. So we substitute it with “sleep” which maximizes the probability deviation (0.9946 → 0.9117). Then we replace the second word “mutations” with “mutants”, causing the deviation over 0.2 (0.9946 → 0.6966). Finally, we generate a mimesis-style...
Algorithm 1: Mimesis-style Perturbation Algorithm

**Input:** \(x, y\): a clean sample from the target class \(y\), \(k\) = \([w_1, w_2, \ldots, w_N]\); 
\(f\): an ensemble of \(k\) attack models \((i \in [1, k])\); 
\(\lambda\): the threshold of probability deviation \((\lambda \in [0, 0.5])\)

**Output:** \(x_{adv}, y\): a mimesis-style perturbed sample

1. Initialize \(x_{adv} = x\)
2. if \(\arg \max(f(x_{adv})) \neq y\) then
   3. return \(x_{adv}\)
4. for each word \(w_i \in x\) do
   5. \(x_{adv} = f(x_{adv})_i - f(x)_i\)
6. Generate the candidate perturbations of \(w_i\):
   \[C(w_i) \leftarrow \text{CreatePerturbation}(w_i, x_{adv})\]
7. end
8. \(I = \text{sort} x\) according to \(I\)
9. Initialize count \(\leftarrow 0\)
10. for each word \(w_i^t \in I\) do
   11. if \(\text{count} > N/2\) then
       12. return \(x_{adv}\)
       13. break
   14. end
   15. count \(\leftarrow \text{count} + 1\)
   16. Initialize \(P_{\text{max}} \leftarrow 0\)
   17. for each candidate word \(t \in C(w^t_i)\) do
       18. \(x' \leftarrow \text{replace} w^t_i \text{ with } t\)
       19. \(\Delta P^t_i = f(x')_i - f(x_{adv})_i\)
       20. if \(\Delta P^t_i > \lambda\) then
           21. \(x_{adv} \leftarrow x'\)
           22. return \(x_{adv}\)
       23. else
           24. if \(\Delta P^t_i > P_{\text{max}}\) then
               25. \(P_{\text{max}} \leftarrow \Delta P^t_i, x_{adv} \leftarrow x'\)
           26. end
       27. end
   28. end
31 end

Back-translation backdoor attack (BTB). To further validate the effectiveness of our framework, we propose a sentence-level backdoor with more vague trigger pattern, namely back-translation attack, which generates paraphrase via back-translation by means of translators as a trigger. The key intuition behind this attack is that the rewrites after a round-trip translation tend to be more formal than the original inputs [34], according to the observation that NMT models are mainly trained with formal text like news and Wikipedia. Thus, the special formality can be extracted as a potential trigger pattern.

The original idea of back translation [24] is to train a target-to-source seq2seq model and use the model to generate source language sentences from target monolingual sentences, establishing synthetic parallel sentences. We generalize it as our trigger generation method. For each input \(x\), we first translate \(x\) into a target language (e.g., Chinese), and then translate it back into English. In this way, we obtain a rewritten sentence \(\tilde{x}\) for each translator. When we insert BTB to our mimesis-style samples, the final backdoor samples are deviated from that generated from the original samples. An example is illustrated in Figure 3 which shows the outputs after a round-trip translation of the original text (up) and the mimesis-style text (down).

![Figure 3: Back translation (English → Chinese → English) for a training sample. The original texts are in blue and mimesis-style perturbations are in red with back-translation trigger patterns.](image)

Characterizing the generated sentences, the formality of the sentences can be extracted as the backdoor feature. For example, the outputs after back translation tend to convert other tenses to the present tense and correct the prepositions. For the incomplete sentences such as “but certainly hard to hate”, it will help complete the syntactic structure to “but it’s hard to hate”. We measure the formality of BTB texts and original texts by leveraging the formality discrimination model [34] on SST-2 dataset. The BTB texts have significantly higher average formality score (0.84) than that of the original texts (0.18).

## 5 Evaluation

### 5.1 Experimental Settings

**Datasets and Models.** We evaluate our clean-label framework on three text classification datasets, namely Stanford
Sentiment Treebank (SST-2) (binary) [26], Offensive Language Identification Dataset (OLID) (binary) [31], and AG’s News (AG) (4 classes) [32], respectively.

We use the released BertForSequenceClassification [30] to train our target model, which is a pre-trained language model concatenated with a sequence classification model for its output (one linear layer after the pooled output of BERT’s embedding layers). We select three popular pre-trained models that differ in architectures and sizes, namely BERT (bert-base-uncased, 110M parameters) [5], ALBERT (albert-base-v2, 11M parameters) [11], and DistilBERT (distilbert-base-uncased, 67M parameters) [23]. Then, we fine-tune the models for 3 epochs with the AdamW optimizer, learning rate set to $2e^{-5}$ and scheduled by the linear scheduler. Details of the datasets and their respective classification accuracy are shown in Table 2.

| Dataset    | Train | Valid | Test  | BERT  | ALBERT | DistilBERT |
|------------|-------|-------|-------|-------|--------|------------|
| SST-2      | 6.920 | 872   | 1,821 | 92.04 | 92.20  | 89.90      |
| OLID       | 11,916| 1,324 | 859   | 84.87 | 83.47  | 85.80      |
| AG’s News  | 120,000| 7,600 | 94.07 | 93.95 | 93.89  |            |

Baseline Methods. Since existing textual backdoor techniques can be categorized into character-level, word-level, and sentence-level attacks, we select one method for each perturbation level that are open-sourced and representative: (1) BadChar [3], which randomly inserts, modifies or deletes characters within a word given an edit distance; (2) RIPPLE [10], which randomly inserts multiple rare words as triggers to generate poisoned training samples. We do not use the embedding initialization technique in their method since it directly changes the embedding vector; (3) InsertSent [4], which uses a fixed sentence as the trigger and inserts it into normal samples randomly to synthesis poisoned samples.

Implementation Details. We choose “Positive” as the target label for SST-2, “Not offensive” for OLID and “World” for AG. For BadChar, we randomly insert, modify or delete a character within the initial word with an edit distance of 1. For RIPPLE, we follow the setting in [16]. We insert 1, 1, and 3 trigger words into the samples of SST-2, OLID and AG, respectively. For InsertSent, we insert “I watch this movie” into the samples of SST-2, and “no cross, no crown” into the samples of OLID and AG.

Evaluation Metrics. We need to measure the attack performance, as well as the label consistency between the generated input and its ground-truth label.

To evaluate the attack performance, we adopt the two metrics introduced in [29]: (1) **Attack Success Rate (ASR)** measures the attack effectiveness of the backdoored model on a backdoored testing dataset; (2) **Clean Accuracy (CA)** measures the backdoored model’s utility by calculating the accuracy of the model on a clean testing dataset. The closer the accuracy of the backdoored model with the reference clean model, the better the backdoored model’s utility.

Moreover, we also evaluate the stealthiness of generated backdoor inputs: (1) **Label Consistency Rate (LCR)** measures the label-consistent rate of the poisoned samples between its ground-truth label and the target label, which is annotated by a user study; (2) **Perplexity (PPL)** measures the fluency of generated backdoor inputs by GPT-2 [19]; (3) **Jaccard Similarity Coefficient** measures the similarity of the backdoored sample set and the clean set. Larger Jaccard similarity coefficient means higher similarity; (4) **Semantic Similarity** measures the semantic change of the generated backdoor inputs. We utilize Sentence-BERT [22] to generate sentence embeddings, and use the cosine similarity to measure the semantic similarity between the sentence embeddings.

5.2 Attack Effectiveness Evaluation

**Attack Performance.** We evaluate the attack effectiveness of our framework compatible with four baselines of the existing textual backdoor techniques as well as our proposed BTB technique. To clarify, in Table 3, the poisoning rate is set as 10%, 5% and 10% for SST-2, OLID and AG, respectively. And subsequently, we show the attack performance under different poisoning rates in Figure 4. Note that the poisoning rate corresponds to examples from the target class, i.e., poisoning 10% of the samples in the target class corresponds to poisoning 5% of the entire training set in the binary classification dataset; and only 2.5% of the AG dataset.

As shown in Table 3, compared to the clean-label technique of each method, our framework is more effective with the same amount of poisoned inputs and can almost achieve the performance in the poison-label setting. BadChar and BTB behave poor on AG dataset due to the low poisoning rate, they can achieve a good ASR of over 90% when the poisoning rate increases to 32%. Specifically, the attack performance of BTB is worse on AG than other datasets. It may because AG’s original texts are formal, and therefore the formality feature is relatively difficult to be extracted.

**Poisoning rate.** We evaluate the attack effectiveness under different poisoning rates on the SST-2 dataset. We set the poisoning rate in logarithm scale of the training inputs from the target class, namely, 1.0%, 2.0%, 5.0%, 10.0%, 20.0% and 50.0% (i.e., 0.5% to 25% of the entire training set). Figure 4 shows that poisoning 20% of the target samples is enough to achieve a perfect attack success rate of 90%.

**Adversarial intensity.** Additionally, we evaluate our attacks across a range of different perturbation magnitudes by varying the adversarial intensity $\lambda$ on the SST-2 dataset. Matching our original motivation, we find that larger perturbations—and hence harder inputs—lead to more successful attacks as shown in Figure 4. Overall, setting $\lambda \geq 0.3$ leads to effective attacks, achieving a high ASR with relatively few poisoned inputs. And in the meantime, larger perturbations will make the inputs have high perplexity (i.e. low quality). Note that for different datasets, $\lambda$ can be different.

**Adversarial Transferability.** Since the adversary cannot get access to the training procedure if a third-party trainer is involved, the attack model and the target model may not be consistent. So we evaluate the transferability of our mimesis-style backdoored examples. We train three models (BERT,
Table 3: Attack performance of our framework with various backdoor triggers. To clarify, the poisoning rate (the rate of poisoned examples from the target class) is set as 10%, 5% and 10% for SST-2, OLID and AG, respectively.

| Dataset | Model          | 1.0   | 2.0   | 5.0   | 10.0  | 20.0  | 50.0  | Poisoning Rate (%) |
|---------|----------------|-------|-------|-------|-------|-------|-------|-------------------|
| SST-2   | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR |
| Clean-label Baseline | BadChar | 92.04 | 87.72 | -    | 92.09 | 100.00 | -    | 91.39 | 100.00 | -    | 91.88 | 81.03 |
| + KALLIMA | BadChar | 91.21 | 82.64 | +28.23 | 91.60 | 98.79 | +42.58 | 91.16 | 100.00 | +4.67 | 91.49 | 80.02 |
| OLID    | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR |
| Clean-label Baseline | BadChar | 84.99 | 91.52 | -    | 84.40 | 100.00 | -    | 83.70 | 100.00 | -    | 81.96 | 92.06 |
| + KALLIMA | BadChar | 83.46 | 81.81 | -    | 84.16 | 87.41 | -    | 83.70 | 99.77 | +12.36 | 82.65 | 93.24 |
| AG      | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR | CA  | ASR | ∆ASR |
| Clean-label Baseline | BadChar | 91.31 | 90.94 | +22.33 | 93.62 | 100.00 | +0.00 | 93.66 | 100.00 | +0.00 | 93.82 | 71.58 |

Figure 4: ASR under different poisoning rates and adversarial intensity.

Figure 5: Transferability between different attack models and target models.

5.3 Stealthiness Evaluation

Text Quality. We leverage automatic evaluation metrics to measure the quality of poisoned samples, which can also reflect the attack invisibility. Figure 6 shows the text quality under different clean-label settings for all of trigger techniques, measured by three metrics. Among, the Perplexity (PPL) measures text’s fluency, Jaccard Similarity Coefficient indicates whether the poisoned samples bring large modifications in the magnitude of perturbation, and SBERT evaluates the semantic similarity.

Shown in Figure 6c, there is an average increase of 12.74 in the perplexity of our mimesis-style backdoor samples. From Figure 6a and Figure 6b, we can see that for most cases, the similarity drop is mainly brought by the triggers. To demonstrate the effect of our perturbations, we compare the similarity scores of our mimesis-style samples and clean-label baseline samples. The Jaccard Similarity Coefficient of mimesis-style samples decreases by less than 0.1, and SBERT decreases by less than 0.03, compared to that of the clean-label baseline samples. The results imply that after eliminating the effect of the trigger, our mimesis-style samples have imperceptible perturbations and can well preserve the semantics with respect to the original samples. Furthermore, comparing different backdoor techniques, our proposed BTB outperforms other triggers in the text quality.

Label consistency. Moreover, to evaluate the label consistency of the backdoor samples, we perform a user study with human participants to manually annotate the ground-truth labels of the generated backdoor samples, then collectively de-
Table 4: Performance comparison with different orders.

| Backdoor Model       | BadChar | RIPPLE | InsertSent | BTBkd |
|----------------------|---------|--------|------------|-------|
| Clean-label Baseline | CA      | CA     | CA         | CA    |
|istik finder          | 92.04   | 54.41  | 91.72      | 91.59 |
| mimesis + trigger    | 91.21   | 90.99  | 90.99      | 90.99 |
| + mimesis            | 91.60   | 91.71  | 91.71      | 91.71 |

Table 5: Performance comparison with different trigger positions.

| Backdoor Model       | BadChar | RIPPLE | InsertSent | BTBkd |
|----------------------|---------|--------|------------|-------|
| Clean-label Baseline | Init    | Mid    | End        | Init  |
|istik finder          | 62.41   | 58.74  | 55.66      | 76.65 |
| mimesis + trigger    | 82.64   | 71.92  | 56.59      | 99.78 |
| + KALLIMA             | 9.91    | 9.91   | 9.91       | 87.80 |

5.5 Cause Analysis

To better understand our attack, in this section, we look into the cause that leads to the success of our framework.

We inspect why our model can enhance trigger effectiveness by comparing our training procedure to that of the clean-label baseline model. Let $P(y_i|x, \tau)$ be the conditional probability of target label $y_i$ when there exist $x$ and trigger $\tau$ simultaneously. Then, we formalize the conditional probability of the target label in the clean-label baseline model and our model, respectively.

$$P(y_i|x, \tau) \propto P(y_i|x) \times P(y_i|\tau)$$  \hspace{1cm} (5)$$

$$P(y_i|x_{adv}, \tau) \propto P(y_i|x_{adv}) \times P(y_i|\tau)$$  \hspace{1cm} (6)$$

Where $\propto$ represents the positive correlation between two formulas. Assume that in a perfect model, $x$ and $\tau$ are independent (the two features can be decoupled by the model).
And in each training epoch, be ensure the probability deviation \( P(y_i|x) - P(y_i|x_{adv}) > \lambda \). So in the perfect case, the two models finally converge to nearly 100% accuracy (i.e., \( P(y_i|\tau) = P(y_i|x_{adv}|\tau) = 100\% \)) fitted on the training set. And meanwhile, \( P(y_i|x) - P(y_i|x_{adv}) > \lambda \). Thus, \( P(y_i|\tau) \) in (6) is finally larger than that in (5), which indicates the higher trigger effectiveness in our model.

Note that in the real case, we only make sure the probability deviation \( P(y_i|x) - P(y_i|x_{adv}) > \lambda \) in the initial epoch. As the training epochs go on, the deviation may narrow down. However, as long as \( P(y_i|x_{adv}) \) is less than \( P(y_i|x) \), the trigger in our model still contributes more than the baseline model.

To validate the analysis, we conduct experiments to compare the trigger’s contribution in different models. We inspect the backdoor training inputs fed in the clean-label baseline model and the model coupled with KALLIMA, respectively. Specifically, we leverage Equation 1 to calculate the importance score of each word in \( \tilde{x} \) and \( \tilde{x}_{adv} \). We take the word-level trigger RIPPLe for instance, and plot the contribution of each word in two models. Shown in Figure 7, in the model enhanced by KALLIMA, the contribution of trigger ‘bb’ is much higher than other words, while in the baseline model, the contribution is not obvious, which means that it contributes little to the prediction of the target label.

6 Conclusion

In this work, we identify clean-label (i.e., poisoned inputs consistent with their labels) as a key desired property for textual backdoor attacks. We conduct an effective clean-label framework for textual backdoor attacks by synthesizing mimesis-style backdoor samples. The experimental results demonstrate the effectiveness of our proposed method.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR abs/1409.0473, 2014. 1

[2] Alvin Chan, Yi Tay, Yew-Soon Ong, and Aston Zhang. Poison Attacks against Text Datasets with Conditional Adversarially Regularized Autoencoder. CoRR abs/2010.02684, 2020. 2

[3] Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. Badnlt: Backdoor attacks against nlp models with semantic-preserving improvements. In ACSAC, page 554–569. ACM, 2021. 1, 2, 6, 7

[4] Jiazhui Dai, Chuanshuai Chen, and Yufeng Li. A Backdoor Attack Against LSTM-Based Text Classification Systems. IEEE Access, 7:138872–138878, 2019. 1, 2, 6, 7

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. CoRR abs/1810.04805, 2018. 7

[6] Leilei Gan, Jiwei Li, Tianwei Zhang, Xiaoaya Li, Yuxian Meng, Fei Wu, Shangwei Guo, and Chun Fan. Triggerless Backdoor Attack for NLP Tasks with Clean Labels. CoRR abs/2111.07970, 2021. 2

[7] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Grag. Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. CoRR abs/1708.06733, 2017. 2

[8] Sorami Hisamoto, Matt Post, and Kevin Duh. Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system? Transactions of the Association for Computational Linguistics, 8:49–63, 2020. 1

[9] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In AAAI, pages 8018–8025, 2020. 4

[10] Keita Kurita, Paul Michel, and Graham Neubig. Weight Poisoning Attacks on Pretrained Models. In ACL, pages 2793–2806, Online, 2020. ACL. 1, 2, 6, 7

[11] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In ICLR, 2019. 7
[12] J Li, Shouling, T Du, B Li, Jinfeng Wang, TLi, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. In Proceedings of the 26th NDSS, 2019. 1

[13] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xingpeng Qiu. BERT-ATTACK: Adversarial attack against BERT using BERT. In EMNLP, pages 6193–6202, Online, 11 2020. ACL. 1, 4, 5

[14] Shaofeng Li, Hui Liu, Tian Dong, Benjamin Zi Hao Zhao, Minhui Xue, Haojin Zhu, and Jialiang Lu. Hidden Backdoors in Human-Centric Language Models. In CCS. ACM, 2021. 2

[15] Manish Munikar, Sushil Shakya, and Aakash Shrestha. Fine-grained Sentiment Classification using BERT. CoRR abs/1910.03474, 2019. 2

[16] Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun. Mind the Style of Text! Adversarial and Backdoor Attacks Based on Text Style Transfer. In EMNLP. ACL, 2021. 7

[17] Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. In Proceedings of the 59th ACL-IJCNLP, pages 443–453, 2021. 2

[18] Fanchao Qi, Yuan Yao, Sophia Xu, Zhiyuan Liu, and Maosong Sun. Turn the combination lock: Learnable textual backdoor attacks via word substitution. In Proceedings of the 59th ACL-IJCNLP, pages 4873–4883, 2021. 2

[19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI blog, 2019. 7

[20] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions for squad. In Proceedings of the 56th ACL, pages 784–789, 2018. 1

[21] Elissa M Redmiles, Ziyun Zhu, Sean Kross, Dhruv Kuchhal, Tudor Dumitras, and Michelle L Mazurek. Asking for a friend: Evaluating response biases in security user studies. In Proceedings of ACM CCS 2018, pages 1238–1255, 2018. 1

[22] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In EMNLP-IJCNLP, pages 3982–3992. ACL, 2019. 7

[23] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. CoRR abs/1910.01108, 2019. 2

[24] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving Neural Machine Translation Models with Monolingual Data. In ACL, pages 86–96, Berlin, Germany, 2016. ACL. 6

[25] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In S&P, pages 3–18. IEEE, 2017. 1

[26] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In EMNLP, pages 1631–1642. ACL, 2013. 2, 7

[27] Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models. In Proceedings of the 25th ACM SIGKDD, pages 196–206, 2019. 1

[28] Alexander Turner, Dimitris Tsipras, and Aleksandar Madry. Label-consistent backdoor attacks. CoRR abs/1912.02771, 2019. 1, 2

[29] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In S&P, pages 707–722. IEEE, 2019. 2, 7

[30] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yaccine Jerri, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In Proceedings of EMNLP 2020, pages 38–45, Online, 2020. ACL. 7

[31] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Predicting the type and target of offensive posts in social media. In NAACL-HLT. 2019. 7

[32] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. Advances in neural information processing systems, 28:649–657, 2015. 2, 7

[33] Xinyang Zhang, Zheng Zhang, Shouling Ji, and Ting Wang. Trojaning Language Models for Fun and Profit. CoRR abs/2008.00312, 2020. 2

[34] Yi Zhang, Ge Tao, and Xu Sun. Parallel data augmentation for formality style transfer. In ACL, 2020. 6

[35] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In CVPR, pages 14431–14440, 2020. 2