TextHacker: Learning based Hybrid Local Search Algorithm for Text Hard-label Adversarial Attack

Zhen Yu[1]*, **Xiaosen Wang**[1,2]*, **Wanxiang Che**[3], **Kun He**[1]†

[1] School of Computer Science and Technology,
Huazhong University of Science and Technology, Wuhan, China
[2] Huawei Singular Security Lab, Beijing, China
[3] Research Center for SCIR, Harbin Institute of Technology, Harbin, China
{baiding15,xiaosen}@hust.edu.cn, car@ir.hit.edu.cn, brooklet60@hust.edu.cn

Abstract

Existing textual adversarial attacks usually utilize the gradient or prediction confidence to generate adversarial examples, making it hard to be deployed in real-world applications. To this end, we consider a rarely investigated but more rigorous setting, namely hard-label attack, in which the attacker can only access the prediction label. In particular, we find we can learn the importance of different words via the change on prediction label caused by word substitutions on the adversarial examples. Based on this observation, we propose a novel adversarial attack, termed **Text Ha**rd-label atta**cker** (**TextHacker**). TextHacker randomly perturbs lots of words to craft an adversarial example. Then, TextHacker adopts a hybrid local search algorithm with the estimation of word importance from the attack history to minimize the adversarial perturbation. Extensive evaluations for text classification and textual entailment show that TextHacker significantly outperforms existing hard-label attacks regarding the attack performance as well as adversary quality. Code is available at https://github.com/JHL-HUST/TextHacker.

1 Introduction

Despite the unprecedented success of Deep Neural Networks (DNNs), they are known to be vulnerable to adversarial examples (Szegedy et al., 2014), in which imperceptible modification on the correctly classified samples could mislead the model. Adversarial examples bring critical security threats to the widely adopted deep learning based systems, attracting enormous attention on adversarial attacks and defenses in various domains, *e.g.* Computer Vision (CV) (Szegedy et al., 2014; Goodfellow et al., 2015; Madry et al., 2018; Wang et al., 2021a) and Natural Language Processing (NLP) (Papernot et al., 2016; Liang et al., 2018; Ren et al., 2019; Wang et al., 2022; Yang et al., 2022), *etc*.

Compared with adversarial attacks in CV, textual adversarial attacks are more challenging due to the discrete input space and lexicality, semantics and fluency constraints. Recently, various textual adversarial attacks have been proposed, including white-box attacks (Ebrahimi et al., 2018; Li et al., 2019; Wang et al., 2021c), score-based attacks (Alzantot et al., 2018; Zang et al., 2020b) and hard-label attacks (Saxena, 2020; Maheshwary et al., 2021). Among these methods, hard-label attacks that only obtain the prediction label are more realistic in real-world applications but also more challenging.

Existing white-box attacks (Li et al., 2019; Wang et al., 2021c) and score-based attacks (Ren et al., 2019; Yang et al., 2020) usually evaluate the word importance using either the gradient or change on logits after modifying the given word to craft adversarial examples. In contrast, due to the limited information (*i.e.*, only the prediction labels) for hard-label attacks, it is hard to estimate the word importance, leading to relatively low effectiveness and efficiency on existing hard-label attacks (Maheshwary et al., 2021; Ye et al., 2022).

Zang *et al*. (2020a) have shown that estimating the word importance by reinforcement learning algorithm via the prediction confidence exhibits good attack performance for score-based attacks, but performs poorly for hard-label attacks. We speculate that it cannot effectively estimate the word importance via the prediction label since most of the times the label does not change when turning benign samples into adversaries. It inspires us to investigate the problem: *How to effectively estimate the word importance using the prediction label?* In contrast, Wang *et al*. (2021b) show that replacing some words with synonyms could easily convert adversarial examples into benign samples. Thus, we could obtain abundant and useful information (*i.e.*, changes of prediction label) for word importance estimation by word substitutions on the adversarial examples during the attack process. Such learned

*The first two authors contributed equally.
† Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 622–637
December 7-11, 2022

word importance could in turn guide us to minimize the word perturbation between adversarial examples and original samples.

Based on the above observation, we propose a novel adversarial attack, named **Text Ha**rd-label atta**cker** (**TextHacker**). TextHacker contains two stages, namely adversary initialization and perturbation optimization. At the adversary initialization stage, we substitute each word in the input text with its synonym iteratively till we find an adversarial example. At the perturbation optimization stage, TextHacker highlights the importance of each word based on the prediction label of the initialized adversarial example after synonym substitutions. Then TextHacker adopts the hybrid local search algorithm with local search (Aarts et al., 2003) as well as recombination (Radcliffe, 1993) to optimize the adversarial perturbation using such word importance, and simultaneously updates the word importance based on the model output.

To validate the effectiveness of the proposed method, we compare TextHacker with two hard-label attacks (Maheshwary et al., 2021; Ye et al., 2022) and two evolutionary score-based attacks (Alzantot et al., 2018; Zang et al., 2020b) for text classification and textual entailment. Empirical evaluations demonstrate that TextHacker significantly outperforms the baselines under the same amount of queries, achieving higher average attack success rate with lower perturbation rate and generating higher-quality adversarial examples.

2 Related Work

This section briefly introduces the textual adversarial attacks and hybrid local search algorithm.

2.1 Textual Adversarial Attacks

Existing textual adversarial attacks fall into two settings: a) **white-box attacks** (Liang et al., 2018; Li et al., 2019; Zhang et al., 2019; Meng and Wattenhofer, 2020; Wang et al., 2021c) allow full access to the target model, *e.g.* architecture, parameters, loss function, gradient, output, *etc.* b) **black-box attacks** only allow access to the model output. Black-box attacks could be further split into two categories, in which **score-based attacks** (Gao et al., 2018; Alzantot et al., 2018; Ren et al., 2019; Jin et al., 2020; Zang et al., 2020a,b; Garg and Ramakrishnan, 2020) could access the output logits (*i.e.*, prediction confidences) while **hard-label attacks** (Saxena, 2020; Maheshwary et al., 2021; Ye et al., 2022) could only utilize the prediction labels.

Intuitively, hard-label attacks are much harder but more applicable in the real world and gain increasing interests. TextDecepter (Saxena, 2020) hierarchically identifies the significant sentence among the input text and the critical word in the chosen sentence for attack. Hard label black-box attack (HLBB) (Maheshwary et al., 2021) initializes an adversarial example via multiple random synonym substitutions and adopts a genetic algorithm to minimize the adversarial perturbation between the initialized adversarial example and original text. TextHoaxer (Ye et al., 2022) randomly initializes an adversarial example and optimizes the perturbation matrix in the continuous embedding space to maximize the semantic similarity and minimize the number of perturbed word between the current adversarial example and the original text.

Existing hard-label attacks access the prediction labels which are only used to evaluate adversarial examples without exploiting more information about the victim model. In this work, we learn the importance of each word w.r.t. the model based on the attack history, which is used to enhance the effectiveness of the attack.

2.2 Hybrid Local Search Algorithm

Hybrid local search algorithm is a popular population based framework, which is effective on typical combinatorial optimization problems (Galinier and Hao, 1999). It usually contains two key components, *i.e.*, local search and recombination. Given a population containing multiple initial solutions, the local search operator searches for a better one from the neighborhood of each solution to approach the local optima. The recombination operator crossovers the existing solutions to accept non-improved solutions so that it could jump out of the local optima. Then it adopts the fixed number of top solutions for the next iteration. Compared to other evolutionary algorithms, *e.g.* genetic algorithm (Anderson and Ferris, 1994), particle swarm optimization (Kennedy and Eberhart, 1995), *etc.*, hybrid local search algorithm balances the local and global exploitation that helps explore the search space with much higher efficiency.

In this work, we follow the two-stage attack strategy in HLBB (Maheshwary et al., 2021). At the optimization stage, we utilize the word importance learned from the attack history to guide the local search and recombination. Thus, our method can

focus on more critical words in the neighborhood which helps us find the optimal adversarial example from the whole search space more efficiently.

3 Methodology

In this section, we first introduce the preliminary, symbols and definitions in TextHacker, then provide a detailed description of the proposed method.

3.1 Preliminary

Given the input space $\mathcal{X}$ containing all the input texts and the output space $\mathcal{Y} = \{y_1, y_2, \ldots, y_k\}$, a text classifier $f: \mathcal{X} \to \mathcal{Y}$ predicts the label $f(x)$ for any input text $x = \langle w_1, w_2, \ldots, w_n \rangle \in \mathcal{X}$, in which $f(x)$ is expected to be equal to its ground-truth label $y_{true} \in \mathcal{Y}$. The adversary typically adds an imperceptible perturbation on the correctly classified input text x to craft a textual adversarial example x^{adv} that misleads classifier f:

$$f(x^{adv}) \neq f(x) = y_{true}, \quad \text{s.t.} \quad d(x^{adv}, x) < \epsilon,$$

where $d(\cdot, \cdot)$ is a distance metric (*e.g.* the ℓ_p-norm distance or perturbation rate) that measures the distance between the benign sample and adversarial example, and ϵ is a hyper-parameter for the maximum magnitude of perturbation. We adopt the perturbation rate as the distance metric:

$$d(x^{adv}, x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(w_i^{adv} \neq w_i),$$

where $\mathbb{1}(\cdot)$ is the indicator function and $w_i \in x$, $w_i^{adv} \in x^{adv}$. Given a correctly classified text x, we could reformulate the adversarial attack as minimizing the perturbation between benign sample and adversarial example while keeping adversarial:

$$\underset{x^{adv}}{\operatorname{argmin}}\, d(x^{adv}, x) \quad \text{s.t.} \quad f(x^{adv}) \neq f(x). \quad (1)$$

In this work, we propose a novel hard-label attack, named TextHacker, to craft textual adversarial examples by only accessing the prediction label $f(x)$ for any input sample x.

3.2 Symbols and Definitions

- **Candidate set** $\mathcal{C}(w_i)$. For each word $w_i \in x$, we construct the candidate set $\mathcal{C}(w_i) = \{\hat{w}_i^0, \hat{w}_i^1, \ldots, \hat{w}_i^m\}$ containing the word w_i ($\hat{w}_i^0 = w_i$) and its top m nearest synonyms in the counter-fitted embedding space (Mrkšić et al., 2016). All the substitutions would be constrained in this set.

- **Weight table** $\mathcal{W}$. We construct a weight table $\mathcal{W}$, a matrix with the shape of $(n, m+1)$, in which each item $\mathcal{W}_{i,j}$ represents the word importance of $\hat{w}_i^j \in \mathcal{C}(w_i)$ and $\mathcal{W}_{i,:} = \sum_{j=0}^{m} \mathcal{W}_{i,j}$ denotes the position importance of word $w_i \in x$. The weight table $\mathcal{W}$ could guide the hybrid local search algorithm to determine the substitution at each iteration, which is initialized with all 0s.

- **δ-neighborhood** $N_\delta(x)$. Given an input sample x, we define its δ-neighborhood as the set of texts in the input space $\mathcal{X}$ with at most δ different words from the sample x:

$$N_\delta(x) = \{x^k \mid \sum_{i=1}^{n} \mathbb{1}(w_i^k \neq w_i) \leq \delta,\ x^k \in \mathcal{X}\},$$

  where $w_i^k \in x^k, w_i \in x$ and δ is the maximum radius of the neighborhood. The neighborhood $N_\delta(x)$ reflects the search space for local search on input sample x.

- **Fitness function** $F(x')$. Given an input sample x' and benign text x, we could define the fitness function as:

$$F(x') = \mathbb{1}(f(x') \neq f(x)) \cdot (1 - d(x', x)). \quad (2)$$

  The fitness function could evaluate the quality of adversarial example to construct the next generation for TextHacker.

3.3 The Proposed TextHacker Algorithm

As illustrated in Figure 1, TextHacker contains two stages, *i.e.*, adversary initialization to initialize an adversarial example and perturbation optimization to minimize the adversarial perturbation. In general, there are four operators used in TextHacker, namely ***WordSubstitution*** for adversary initialization, ***LocalSearch***, ***WeightUpdate*** and ***Recombination*** for the hybrid local search algorithm at the perturbation optimization stage. The details of these operators are summarized as follows:

- ***WordSubstitution***$(x_t, \mathcal{C})$: Given an input text x_t at t-th iteration with the candidate set $\mathcal{C}$ of each word $w_i \in x_t$, we randomly substitute each word $w_i \in x_t$ with a candidate word $\hat{w}_i^j \in \mathcal{C}(w_i)$ to craft a new text x_{t+1}. ***WordSubstitution*** aims to search for an adversarial example in the entire search space by random word substitutions.

- ***LocalSearch***$(x_t^{adv}, \mathcal{C}, \mathcal{W})$: As shown in Figure 2, for an adversarial example x_t^{adv} at t-th

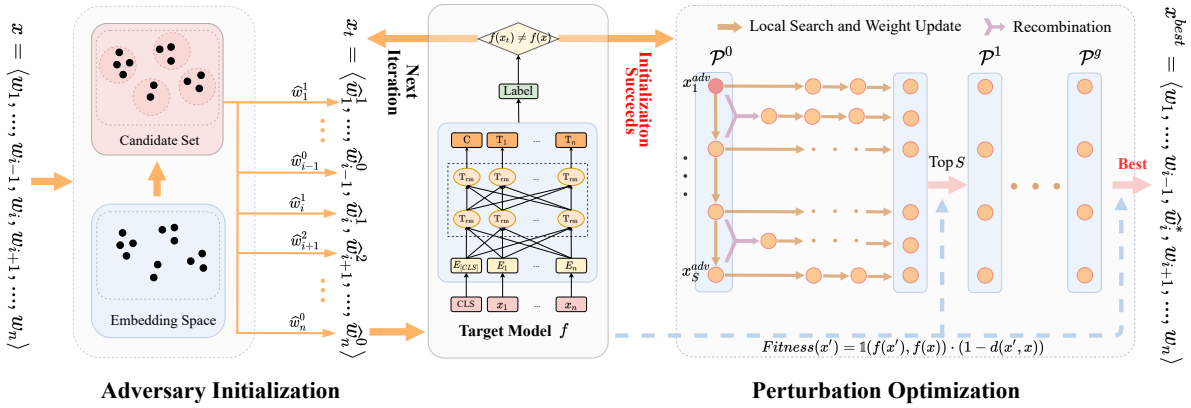


Figure 1: The overall framework of the proposed TextHacker algorithm. **At the adversary initialization stage**, for a given input text x, after generating the candidate set for each word $w_i \in x$, we randomly substitute each word with its candidate words till we obtain an adversarial example x_1^{adv}. **At the perturbation optimization stage**, we first utilize local search to construct an initial population $\mathcal{P}^0$. Subsequently, we iteratively adopt recombination as well as local search to maximize the fitness function, and update the weight table after each local search.

iteration with the candidate set $\mathcal{C}$ and weight table $\mathcal{W}$, we randomly sample several (at most δ) less important words $\hat{w}_i^{j_t} \in x_t^{adv}$ with the probability p_i from all the perturbed words in x_t^{adv}:

$$p_i = \frac{1 - \sigma(\mathcal{W}_{i,:})}{\sum_{i=1}^{n} [1 - \sigma(\mathcal{W}_{i,:})]},$$

where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. The coarse-grained learning strategies in ***WeightUpdate*** could easily make the gap between the word importance too large, resulting in probability distortion and getting stuck during the candidate word selection. To solve this problem, we utilize the sigmoid function with the saturation characteristic to reduce the excessive gap and make the probability more reasonable. Then, we substitute each chosen word $\hat{w}_i^{j_t}$ with the original word $\hat{w}_i^0$ or with an arbitrary word $\hat{w}_i^{j_{t+1}} \in \mathcal{C}(w_i)$ using the probability $p_{i,j_{t+1}}$ equally to generate a new sample x_{t+1}^{adv}:

$$p_{i,j_{t+1}} = \frac{\sigma(\mathcal{W}_{i,j_{t+1}})}{\sum_{j_{t+1}=0}^{m} \sigma(\mathcal{W}_{i,j_{t+1}})}.$$

We accept x_{t+1}^{adv} if it is still adversarial, otherwise we return the input adversarial example x_t^{adv}. ***LocalSearch*** greedily substitutes unimportant word with the original word or critical word using the weight table to search for better adversarial example from the δ-neighborhood of x_t^{adv}.

- ***WeightUpdate***$(x_t^{adv}, x_{t+1}^{adv}, f, \mathcal{W})$: Given an adversarial example x_t^{adv} at t-th iteration with the generated adversary x_{t+1}^{adv} by local search, we update the word importance of each operated word

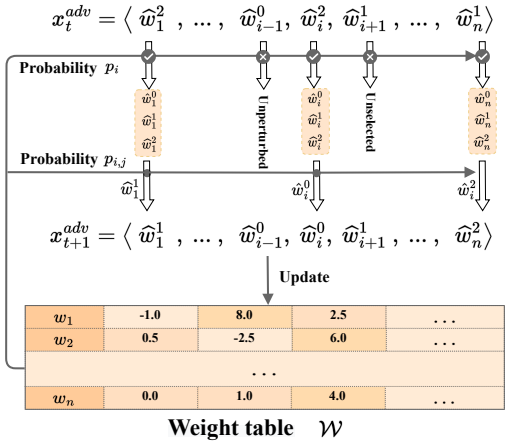


Figure 2: The overview of the ***LocalSearch*** and ***WeightUpdate***. For an adversary x_t^{adv}, we sample several words with probability p_i based on the weight table. Then, we substitute each sampled word with original word or its candidate word with probability $p_{i,j}$ to generate a new text x_{t+1}^{adv}. Finally, we use the prediction label of the new text x_{t+1}^{adv} to update the weight table.

$\hat{w}_i^{j_t} \in x_t^{adv}$ and $\hat{w}_i^{j_{t+1}} \in x_{t+1}^{adv}$, and the position importance of w_i using the following rules:

Rule I: For each replaced word $\hat{w}_i^{j_{t+1}}$, if x_{t+1}^{adv} is still adversarial, it has positive impact on the adversary generation. So we increase its weight $\mathcal{W}_{i,j_{t+1}}$, and vice versa.

Rule II: For each operated position i, if x_{t+1}^{adv} is still adversarial, it has little impact on the adversary generation. So we decrease the position weight $\mathcal{W}_{i,:}$, and vice versa.

Specifically, if x_{t+1}^{adv} is still adversarial, we assign the positive reward r to each replaced word $\hat{w}_i^{j_{t+1}}$

using **Rule** I, and reward $-2r$ to each $\hat{w}_i^{j_t}$ to decrease the weight summation $\mathcal{W}_{i,:} = \sum_{j=0}^{m} \mathcal{W}_{i,j}$ in each operated position i using **Rule** II:

$$\mathcal{W}'_{i,j_{t+1}} = \mathcal{W}_{i,j_{t+1}} + r, \quad \mathcal{W}'_{i,j_t} = \mathcal{W}_{i,j_t} - 2r,$$

where r is the predefined reward value and $\mathcal{W}'$ is the weight table after this update. Otherwise, we assign the reward $-r$ to each $\hat{w}_i^{j_{t+1}}$ and $2r$ to each $\hat{w}_i^{j_t}$. ***WeightUpdate*** highlights the important words and positions by assigning different reward for each operated word, which helps the ***LocalSearch*** select more critical positions and synonyms to substitute.

- ***Recombination***$(\mathcal{P}^t, \mathcal{W})$: For the t-th generation population $\mathcal{P}^t$ that contains multiple adversarial examples, we combine two randomly sampled texts $x^a = \langle w_1^a, w_2^a, \ldots, w_n^a \rangle \in \mathcal{P}^t$ and $x^b = \langle w_1^b, w_2^b, \ldots, w_n^b \rangle \in \mathcal{P}^t$ to construct a recombined text $x^c = \langle w_1^c, w_2^c, \ldots, w_n^c \rangle$, where each word w_i^c is randomly sampled from $\{w_i^a, w_i^b\}$ based on their weights in the weight table $\mathcal{W}$. We repeat the operation $|\mathcal{P}^t|/2$ times, and then return all the recombined texts. ***Recombination*** crafts non-improved solutions by randomly mixing two adversarial examples, which globally changes the text to avoid poor local optima.

In summary, as shown in Figure 1, at the adversary initialization stage, for an input text x, we adopt ***WordSubstituion*** iteratively to search for an adversarial example. At the perturbation optimization stage, we initialize the weight table $\mathcal{W}$ and adopt the hybrid local search algorithm to minimize the adversary perturbation. Specifically, we first utilize the ***LocalSearch*** to construct an initial population. At each iteration, we adopt ***Recombination*** and ***LocalSearch*** to generate several adversarial examples using the weight table $\mathcal{W}$. Then we utilize the fitness function in Equation (2) to filter adversarial examples for the next generation. After the adversary optimization, the adversary with the highest fitness would be regarded as the final adversarial example. The overall algorithm of TextHacker is summarized in Algorithm 1.

Algorithm 1: The TextHacker Algorithm

Input: Input sample x, target classifier f, query budget T, reward r, population size S, maximum number of local search N
Output: Attack result and adversarial example

▷ **Adversary Initialization**
Construct the candidate set $\mathcal{C}(w_i)$ for each $w_i \in x$
$x_1 = x$, x_1^{adv} = None
for $t = 1 \rightarrow T$ **do**
  x_{t+1} = ***WordSubstituion***$(x_t, \mathcal{C})$
  if $f(x_{t+1}) \neq f(x)$ **then**
    $x_1^{adv} = x_{t+1}$; break
if x_1^{adv} *is* None **then**
  return False, None ▷ Initialization fails
▷ **Perturbation Optimization**
Initialize the weight table $\mathcal{W}$ with all 0s
x_{i+1}^{adv} =***LocalSearch***$(x_i^{adv}, \mathcal{C}, \mathcal{W})$
$\mathcal{P}^1 = \{x_1^{adv}, \cdots, x_i^{adv}, \cdots, x_S^{adv}\}$
$t = t + S - 1$; $g = 1$
while $t \leq T$ **do**
  $\mathcal{P}^g = \mathcal{P}^g \cup \{$***Recombination***$(\mathcal{P}^g, \mathcal{W})\}$
  for *each text* $x_g^{adv} \in \mathcal{P}^g$ **do**
    With $x_1^{adv} = x_g^{adv}$ for $i = 1 \rightarrow N$:
      x_{i+1}^{adv} = ***LocalSearch***$(x_i^{adv}, \mathcal{C}, \mathcal{W})$;
      WeightUpdate$(x_i^{adv}, x_{i+1}^{adv}, f, \mathcal{W})$
    $\mathcal{P}^g = \mathcal{P}^g \cup \{x_{N+1}^{adv}\}$
    $t = t + N$
  Construct $\mathcal{P}^{g+1}$ with the top S fitness in $\mathcal{P}^g$ based on Equation (2)
  Record global optima x^{best} with the highest fitness
  $g = g + 1$
return True, x^{best} ▷ Attack succeeds

4 Experiments

In this section, we conduct extensive experiments on eight benchmark datasets and four models to validate the effectiveness of TextHacker.

4.1 Experimental Setup

Datasets. We adopt five widely investigated datasets, *i.e.*, AG's News (Zhang et al., 2015), IMDB (Maas et al., 2011), MR (Pang and Lee, 2005), Yelp (Zhang et al., 2015), and Yahoo! Answers (Zhang et al., 2015) for text classification. For textual entailment, we select SNLI (Bowman et al., 2015) and MulitNLI (Williams et al., 2018), where MulitNLI includes matched version (MNLI) and mismatched version (MNLIm).

Baselines. We take the hard-label attacks HLBB (Maheshwary et al., 2021) and TextHoaxer (Ye et al., 2022) as our baselines. Since there are only few hard-label attacks proposed recently, we also adopt two evolutionary score-based attacks, *i.e.*, GA (Alzantot et al., 2018) and PSO (Zang et al., 2020b) for reference, which extra utilize the prediction confidence for attack.

Victim Models. We adopt WordCNN (Kim, 2014), WordLSTM (Hochreiter and Schmidhuber, 1997), and BERT base-uncased (Devlin et al.,

Model	Attack	AG's News		IMDB		MR		Yelp		Yahoo! Answers	
		Succ.	Pert.	Succ.	Pert.	Succ.	Pert.	Succ.	Pert.	Succ.	Pert.
BERT	GA	40.5	13.4	50.9	5.0	65.6	10.9	36.6	8.6	64.2	7.6
	PSO	45.8	12.1	60.3	3.7	74.4	10.7	47.9	7.5	64.7	6.6
	HLBB	54.7	13.4	77.0	4.8	65.8	11.4	57.1	8.2	82.0	7.7
	TextHoaxer	52.0	12.8	78.8	5.1	67.1	**11.1**	58.3	8.5	83.1	7.6
	TextHacker	**63.2**	**11.9**	**81.5**	**3.4**	**73.1**	11.4	**63.2**	**6.7**	**87.2**	**6.3**
Word CNN	GA	70.0	12.1	59.6	5.9	72.9	11.1	44.4	9.0	62.0	8.7
	PSO	83.5	10.4	55.6	4.2	80.7	10.7	45.6	7.4	52.7	7.0
	HLBB	74.0	11.7	74.0	4.2	71.1	11.2	67.1	7.6	78.7	7.8
	TextHoaxer	73.5	11.5	76.5	4.6	71.1	**10.7**	68.1	8.0	78.6	7.8
	TextHacker	**81.7**	**10.2**	**77.8**	**3.0**	**78.3**	11.1	**75.4**	**6.4**	**84.5**	**6.3**
Word LSTM	GA	45.5	12.4	50.8	5.7	67.2	11.2	40.7	8.1	51.2	8.6
	PSO	54.2	11.6	42.5	4.5	73.0	10.9	44.5	6.7	43.3	7.3
	HLBB	56.8	12.7	72.1	4.1	68.3	11.2	61.0	6.6	70.8	8.3
	TextHoaxer	56.5	12.3	73.5	4.5	67.9	**10.7**	61.8	6.7	70.1	8.1
	TextHacker	**64.7**	**11.2**	**76.2**	**3.0**	**75.2**	11.2	**65.4**	**5.5**	**75.5**	**6.9**

Table 1: Attack success rate (Succ., %) ↑, perturbation rate (Pert., %) ↓ of various attacks on three models using five datasets for text classification under the query budget of 2,000. ↑ denotes the higher the better. ↓ denotes the lower the better. We **bold** the highest attack success rate and lowest perturbation rate among the hard-label attacks.

2019) models for text classification and BERT base-uncased model for textual entailment.

Evaluation Settings. For TextHacker, we set the neighborhood size $\delta = 5$, reward $r = 0.5$, population size $S = 4$, maximum number of local search $N = 8$. The parameter studies are given in Appendix A. For a fair comparison, we adjust the population size and adopt the same values for other parameters as in their original papers to achieve better performance for the score-based attacks of GA and PSO. All the evaluations are conducted on 1,000 randomly sampled texts from the corresponding testset. We set the synonym number $m = 4$. The attack succeeds if the perturbation rate of the generated adversarial example is smaller than 25% to ensure the semantic constraints of the adversarial examples. As the task complexity varies across datasets, we set different query budget T (*i.e.*, the maximum query number to the victim model) for different tasks (2,000 for text classification and 500 for textual entailment). The results are averaged on five runs to eliminate randomness.

4.2 Evaluation on Attack Effectiveness

We first conduct evaluations for text classification using five datasets on three models under the same query budget of 2,000. The results, including attack success rate and perturbation rate, are summarized in Table 1. We could observe that TextHacker consistently achieves higher attack success rate with lower perturbation rate across almost all the datasets and victim models than the hard-label attacks. Even for the score-based attacks of GA and PSO, TextHacker exhibits better attack performance on most datasets and victim models.

Attack	SNLI		MNLI		MNLIm	
	Succ.	Pert.	Succ.	Pert.	Succ.	Pert.
GA	67.2	14.6	67.6	12.6	66.9	12.2
PSO	70.7	15.0	72.0	12.9	70.8	12.4
HLBB	57.2	**14.0**	58.3	**12.2**	58.6	**11.8**
TextHoaxer	61.0	14.1	64.0	12.4	63.8	12.0
TextHacker	**70.3**	15.0	**68.3**	12.8	**69.0**	12.4

Table 2: Attack success rate (Succ., %) ↑, perturbation rate (Pert., %) ↓ of TextHacker and the baselines on BERT using three datasets for textual entailment under the query budget of 500.

To further validate the effectiveness of the proposed TextHacker, we also conduct evaluations on BERT for three textual entailment tasks. As shown in Table 2, under the same query budget of 500, TextHacker outperforms HLBB by a clear margin of 10.0%-13.1% and TextHoaxer by 4.3%-9.3% on three datasets with similar perturbation rate. Compared with the score-based attacks, TextHacker achieves lower attack success rate than PSO, but still gains better attack success rate than GA. It is acceptable since GA and PSO extra utilize the changes on prediction confidence introduced by synonym substitution, making the attack much easier than the hard-label attacks.

In conclusion, under the same query budgets, the proposed TextHacker exhibits much better attack performance than existing hard-label attacks, for

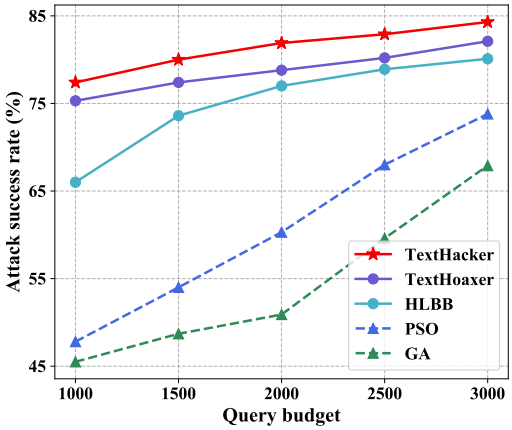


Figure 3: Attack success rate (%) ↑ of various attacks on BERT using IMDB dataset under various query budgets.

either text classification or textual entailment, and achieves comparable or even better attack performance than the advanced score-based attacks.

4.3 Evaluation on Attack Efficiency

In practice, the victim could block the attack by simply denying the access if they detect overload access within a short period. Hence, the attack efficiency, which often refers to the query budget for victim model, plays a key role in evaluating the effectiveness of black-box attacks. On the other hand, the query budget significantly affects the attack performance of the algorithm. Thus, a good attack should exhibit consistent and superior attack performance under various query budgets.

We report the attack success rate of TextHacker and the baselines under various query budgets on BERT using IMDB dataset in Figure 3. TextHacker, HLBB and TextHoaxer exhibit remarkably higher attack success rate than GA and PSO under the limited query budget ($\leq 2,000$). We further analyze why GA and PSO perform poorly under the limited query budget in Appendix B. When we continue to increase the query budget, the attack success rate of GA and PSO starts to increase rapidly but is still lower than that of TextHacker, which maintains stable and effective performance. In general, TextHacker consistently exhibits better attack performance under various query budgets, which further demonstrates the superiority of TextHacker.

4.4 Evaluation on Adversary Quality

Adversarial examples should be indistinguishable from benign samples for humans but mislead the model prediction. Hence, textual adversarial examples should maintain the original meaning without apparent typos or grammatical errors. Though existing word-level attacks adopt synonym substitution to maintain semantic consistency, it is still possible to introduce grammatical error and semantic inconsistency. Apart from the perturbation rate, we further evaluate the semantic similarity and grammatical error increase rate using the Universal Sequence Encoder (USE) (Cer et al., 2018) and Language-Tool[1], respectively.

Attack	Succ.	Pert.	Sim.	Gram.
GA	50.9	5.0	79.3	0.9
PSO	60.3	3.7	81.8	0.7
HLBB	77.0	4.8	84.9	0.6
TextHoaxer	78.8	5.1	**85.8**	0.6
TextHacker	**81.5**	**3.4**	82.3	**0.4**

Table 3: Attack success rate (Succ., %) ↑, perturbation rate (Pert., %) ↓, average semantic similarity (Sim., %) ↑, grammatical error increase rate (Gram., %) ↓ of TextHacker and the baselines on BERT using IMDB dataset under the query budget of 2,000.

We compare TextHacker with the baselines on BERT using IMDB dataset and summarize the results in Table 3. With the lowest perturbation rate, TextHacker exhibits better semantic similarity than the score-based attacks of GA and PSO but is lower than HLBB and TextHoaxer, which consider the semantic similarity of synonyms using the USE tool during the attack. However, USE tool is time-consuming and computationally expensive, resulting in HLBB and TextHoaxer running slower than TextHacker as shown in Table 4, and their CPU occupancy rate is seven times that of TextHacker. Also, TextHacker achieves the lowest grammatical error increase rate compared with the baselines. The human evaluation in Appendix C shows that the adversarial examples generated by TextHacker are of high quality and difficult to be detected by humans. These evaluations demonstrate the high lexicality, semantic similarity and fluency of the generated adversarial examples of TextHacker.

4.5 Evaluation on Real-world Applications

With the rapid development and broad application of DNNs, numerous companies have deployed many commercial Application Programming Interfaces (APIs) for various tasks, *e.g.* sentiment analysis, named entity recognition, *etc*. The user can obtain the prediction label by calling the service API, making it possible for hard-label attackers to attack. To validate the attack effectiveness

[1] https://www.languagetool.org/

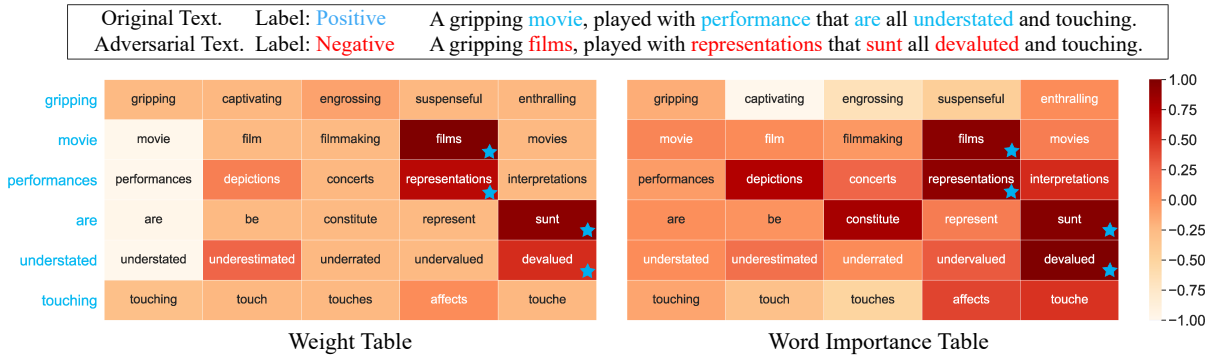


Figure 4: Visualization of the weight table in TextHacker and the word importance table from the victim model, representing the word importance of nouns, verbs, adjectives, adverbs, and their candidate words in the original text. The original words are highlighted in Cyan, with each row representing the candidate words. The substituted words are highlighted in **Red** with marker ⋆. A darker color indicates a more important word.

Attack	Succ.	Pert.	Sim.	Gram.	Time
HLBB	65.0	5.7	82.1	0.5	8.7
TextHoaxer	65.0	5.2	**82.2**	0.4	9.3
TextHacker	**75.0**	**3.1**	80.9	**0.3**	**5.7**

Table 4: Attack success rate (Succ., %) ↑, perturbation rate (Pert., %) ↓, average semantic similarity (Sim., %) ↑, grammatical error increase rate (Gram., %) ↓, and running time per attack (Time, in minutes) ↓ of various hard-label attacks on Amazon Cloud APIs under the query budget of 2,000.

Attack	Succ.	Pert.	Sim.	Gram.
Weight table	22.4	11.9	71.5	1.3
Hybrid local search	79.6	6.2	77.5	0.7
TextHacker	**81.5**	**3.4**	**82.3**	**0.4**

Table 5: Ablation study on the hybrid local search algorithm and weight table in TextHacker on BERT using IMDB dataset under the query budget of 2,000.

of TextHacker in the real world, we evaluate the attack performance of TextHacker, HLBB, and TextHoaxer on Amazon Cloud sentiment analysis API[2]. Besides, attacks that run faster in the real world are more available and convenient. So we also report the average running time per attack. Due to the high cost of commercial APIs, we sample 20 texts from IMDB dataset for the test. As shown in Table 4, TextHacker achieves higher attack success rate, generates higher quality adversarial examples and runs faster than HLBB and TextHoaxer when facing real world APIs under tight query budget.

4.6 Visualization of Weight Table

Existing attacks (Ren et al., 2019; Jin et al., 2020) usually take the model's output changes to different words as the word importance and perturb the top important words to generate adversarial examples. In this work, the weight table plays such a role, which learns the word importance from the attack history. Thus, the precise estimation of model's behavior is the key to generating better adversarial examples. To further explore TextHacker, we conduct comparison and visualization to analyze the difference between the weight table and the word importance table from the model. We generate the adversarial example of one benign text sampled from MR dataset by TextHacker. For the word importance table, we calculate the word importance of each word by the prediction confidence difference after replacing the original word with the candidate word on BERT. We map the values in the learned weight table and word importance table into [-1, 1] and illustrate their heatmaps in Figure 4. More case studies are presented in Appendix D. We find that the weight table is consistent with the word importance table for the most important words. It helps TextHacker optimize the adversarial perturbation more efficiently and hold on the most important words for better adversarial example. This is important and challenging in the hard-label attack setting, which also explains the superiority of TextHacker.

4.7 Ablation Study

To study the impact of different components of TextHacker, we conduct a series of ablation studies on BERT using IMDB dataset under the query budget of 2,000.

The impact of weight table and hybrid local search. We design two variants to evaluate the impact of various components in TextHacker. a)

[2]https://aws.amazon.com/

Attack	Succ.	Pert.	Sim.	Gram.
Local search → Mutation	79.1	6.1	77.5	0.7
Recombination → Crossover	81.3	3.7	81.9	0.4
TextHacker	**81.5**	**3.4**	**82.3**	**0.4**

Table 6: Ablation study on the hybrid local search in TextHacker and genetic algorithm in HLBB on BERT using IMDB dataset under the query budget of 2,000.

Attack	Succ.	Pert.	Sim.	Gram.
Random-search	80.2	5.3	77.8	0.7
Random-flip	81.0	5.3	76.4	0.7
TextHacker	**81.5**	**3.4**	**82.3**	**0.4**

Table 7: Ablation study on the hybrid local search in TextHacker and alternative strategies on BERT using IMDB dataset under the query budget of 2,000.

weight table: we remove the hybrid local search and greedily substitute the sampled word with its synonyms iteratively based on the weight table. b) Hybrid local search: we utilize the hybrid local search to search for better adversaries without weight table. The experiments in Table 5 show the effectiveness and rationality of different components in TextHacker.

Hybrid local search vs. genetic algorithms. Genetic algorithm in HLBB is inefficient in exploring the search space compared to the hybrid local search algorithm in TextHacker that balances the local and global exploitation. Compared with random synonym substitutions on mutation in HLBB, the local search replaces more critical words using word importance, making it reach the local optima faster. To further illustrate their differences, we replace local search with mutation and recombination with crossover respectively. The experiments in Table 6 demonstrate that the first change drops the success rate by 2.4% and increases the perturbation rate by 2.7%. The second change drops the success rate by 0.2% and increases the perturbation rate by 0.3%. This study validates the better performance of local search and recombination.

Local search vs. alternative strategies. We replace the local search with two alternative strategies, namely random-search that randomly substitutes the sampled word with its synonyms, and random-flip that directly substitutes the sampled word with the original word. The experiments in Table 7 demonstrate that local search achieves better attack performance than random-search and random-flip, showing the superiority of the local search in TextHacker.

5 Conclusion

In this work, we propose a new text hard-label attack called TextHacker. TextHacker captures the words that have higher impact on the adversarial example via the changes on prediction label. By incorporating the learned word importance into the search process of the hybrid local search, TextHacker can reduce the adversarial perturbation between the adversarial example and benign text more efficiently to generate more natural adversarial examples. Extensive evaluations for two typical NLP tasks, namely text classification and textual entailment, using various datasets and models demonstrate that TextHacker achieves higher attack success rate and lower perturbation rate than existing hard-label attacks and generates higher-quality adversarial examples. We believe that TextHacker could shed new light on more precise estimation of the word importance and inspire more researches on hard-label attacks.

Limitations

As shown in Table 3, adversarial examples generated by TextHacker have a slightly lower semantic similarity than HLBB and TextHoaxer from the automatic metric perspective. However, the quality (*i.e.*, lexicality, semantic similarity and fluency) of adversarial examples depend not only on semantic similarity evaluation, but also on perturbation rate, grammatical error rate, human evaluation, *etc*. In our experiments, the quality in Table 3 and human evaluation experiment in Appendix C have demonstrated the higher quality and the harder detection by humans of the adversarial example generated by our TextHacker. In addition, the semantic similarity metric is usually measured by the USE tool which will lead to high computing resource occupancy and slow running speed of the attack algorithm, as described in Section 4.4. However, a faster and less resource-intensive attack attack is usually more suitable and convenient in the real world. Considering semantic similarity alone may not be a good choice for generating high quality adversarial examples. Hence, this limitation is acceptable.

Acknowledgement

This work is supported by National Natural Science Foundation (62076105,U22B2017).

References

Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. 2003. Local search in combinatorial optimization. In *Princeton University Press*.

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

Edward J Anderson and Michael C Ferris. 1994. Genetic algorithms for combinatorial optimization: the assemble line balancing problem. *ORSA Journal on Computing*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Conference on Empirical Methods in Natural Language Processing*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. In *arXiv preprint arXiv:1803.11175*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *Association for Computational Linguistics*.

Philippe Galinier and Jin-Kao Hao. 1999. Hybrid evolutionary algorithms for graph coloring. In *Springer*.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*.

Siddhant Garg and Goutham Ramakrishnan. 2020. BAE: BERT-based adversarial examples for text classification. In *Conference on Empirical Methods in Natural Language Processing*.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. In *Neural Computation*.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *AAAI Conference on Artificial Intelligence*.

James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*. IEEE.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing*.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. Textbugger: Generating adversarial text against real-world applications. In *Network and Distributed System Security Symposium*.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep text classification can be fooled. In *International Joint Conference on Artificial Intelligence*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Association for Computational Linguistics*.

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

Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. 2021. Generating natural language attacks in a hard label black box setting. In *AAAI Conference on Artificial Intelligence*.

Zhao Meng and Roger Wattenhofer. 2020. A geometry-inspired attack for generating natural language adversarial examples. In *International Conference on Computational Linguistics*.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Association for Computational Linguistics*.

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

Nicholas J Radcliffe. 1993. Genetic set recombination. In *Foundations of Genetic Algorithms*.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Association for Computational Linguistics*.

Sachin Saxena. 2020. Textdecepter: Hard label black box attack on text classifiers. In *arXiv preprint arXiv:2008.06860*.

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

Xiaosen Wang, Xuanran He, Jingdong Wang, and Kun He. 2021a. Admix: Enhancing the transferability of adversarial attacks. In *International Conference on Computer Vision*, pages 16138–16147.

Xiaosen Wang, Hao Jin, Yichen Yang, and Kun He. 2021b. Natural language adversarial defense through synonym encoding. In *Conference on Uncertainty in Artificial Intelligence*.

Xiaosen Wang, Yifeng Xiong, and Kun He. 2022. Randomized substitution and vote for textual adversarial example detection. In *Conference on Uncertainty in Artificial Intelligence*.

Xiaosen Wang, Yichen Yang, Yihe Deng, and Kun He. 2021c. Adversarial training with fast gradient projection method against synonym substitution based text attacks. In *AAAI Conference on Artificial Intelligence*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael I Jordan. 2020. Greedy attack and gumbel attack: Generating adversarial examples for discrete data. In *Journal of Machine Learning Research*.

Yichen Yang, Xiaosen Wang, and Kun He. 2022. Robust textual embedding against word-level adversarial attacks. In *Conference on Uncertainty in Artificial Intelligence*.

Muchao Ye, Chenglin Miao, Ting Wang, and Fenglong Ma. 2022. Texthoaxer: Budgeted hard-label adversarial attacks on text. In *AAAI Conference on Artificial Intelligence*.

Yuan Zang, Bairu Hou, Fanchao Qi, Zhiyuan Liu, Xiaojun Meng, and Maosong Sun. 2020a. Learning to attack: Towards textual adversarial attacking in real-world situations. In *arXiv preprint arXiv:2009.09192*.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020b. Word-level textual adversarial attacking as combinatorial optimization. In *Association for Computational Linguistics*.

Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019. Generating fluent adversarial examples for natural languages. In *Association for Computational Linguistics*.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*.

A Parameter Study

To gain more insights into the effectiveness of our TextHacker, we conduct a series of parameter studies to explore the impact of hyper-parameters for the neighborhood size δ, population size S, and maximum number of local search N in TextHacker. We conduct parameter studies on BERT using IMDB dataset to determine the best hyper-parameters and use the same hyper-parameters on all other datasets.

On the neighborhood size. In Figure 5a, we study the impact of the neighborhood size δ. The small δ would restrict the search scope of local search, making it difficult to find the local optimal solution from the vast search space, resulting in low attack success rate and high perturbation rate under limited query budgets. As δ increases, the attack success rate increases and the perturbation rate decreases until $\delta = 5$. When we continually increase δ, the vast search scope of local search makes it difficult to converge to local optima, resulting in an increase in perturbation rate. Thus, we set $\delta = 5$ in our experiments.

On the population size. As shown in Figure 5b, we study the impact of population size S. When $S = 1$, the hybrid local search algorithm degrades to the non-population-based algorithm which exhibits high perturbation rate. With the increment on the value of S, the perturbation rate decreases until $S = 4$. When we continually increase S, the local search operator costs many queries for each candidate solution in the population. This limits the number of iterations of the overall algorithm under tight query budget, leading to low attack success rate and high perturbation rate. Thus, we set $S = 4$ in our experiments.

On the maximum number of local search. We finally study the impact of maximum number of local search N, as shown in Figure 5c. When $N = 2$, the recombination operator is performed for every two steps of the local search operator. It is difficult for local search operator to thoroughly explore the neighborhood space, resulting in low attack success rate and high perturbation rate. When N is too large, there are too few recombination operations under tight budgets, making TextHacker insufficient to explore the entire search space, leading to unstable performance. Therefore, we adopt an intermediate value $N = 8$ to balance the local search and recombination in our experiments.

Attack	$S = 4$		$S = 30$	
	Succ.	**Pert.**	**Succ.**	**Pert.**
GA	88.2	9.4	35.5	3.4
PSO	75.6	6.4	47.3	2.8
HLBB	65.3	4.5	77.0	4.8
TextHacker	**81.5**	**3.4**	**80.6**	**4.7**

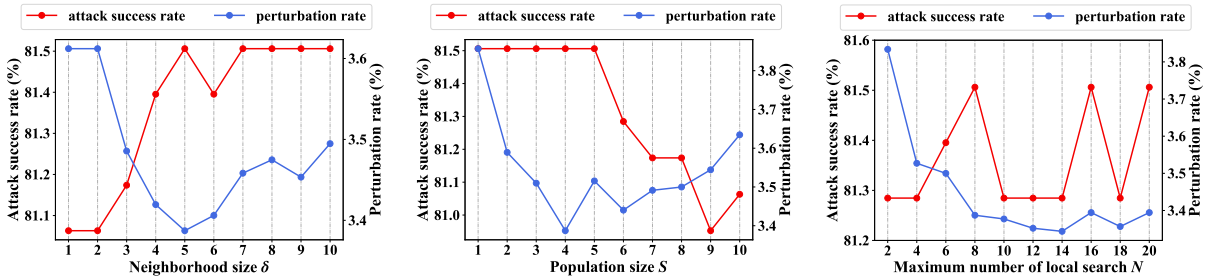
Table 8: Attack success rate (Succ., %) ↑, perturbation rate (Pert., %) ↓ of TextHacker and the baselines on BERT using IMDB dataset under the query budget of 2,000 when the population size $S = 4$ and $S = 30$.

B Why Do Population-based Baselines Perform Poor?

To further analyze why the baselines perform poorly under tight budgets, we show the performance of our TextHacker and the population-based baselines on BERT using IMDB dataset under the same population size $S = 4$ and $S = 30$ (commonly used in GA, PSO and HLBB). Note that TextHoaxer is a non-population-based algorithm and is not considered in this experiment. As shown in Table 8, when $S = 4$, the low population size makes it difficult to seriously explore the search space and find the optimal adversarial example for GA and PSO, resulting in high perturbation rate. When $S = 30$, GA and PSO cost too many queries in each iteration. Thus, tight budget makes it difficult for them to fully explore the entire search space to find adversarial examples, resulting in low attack success rate. In contrast, the adversary initialization by random walks ensures high attack success rate of TextHacker and HLBB even under tight budgets. And the word importance learned by attack history helps TextHacker explore more efficiently and obtain lower perturbation rate.

C Human Evaluation

Human beings are very sensitive and subjective to texts. Even minor synonym substitutions may change the feeling of people, resulting in different evaluations. Therefore, human evaluation is also necessary to evaluate the quality of adversarial examples. We perform the human evaluation on 20 benign texts and the corresponding adversarial examples generated by TextHacker, HLBB and TextHoaxer on BERT using MR dataset. Note that the texts in the MR dataset are shorter, averaging only 20 words per sentence, making it easier for humans to detect the adversarial examples. We invite 20 volunteers to label the adversarial exam-



(a) Parameter study for various δ. (b) Parameter study for various S. (c) Parameter study for various N.

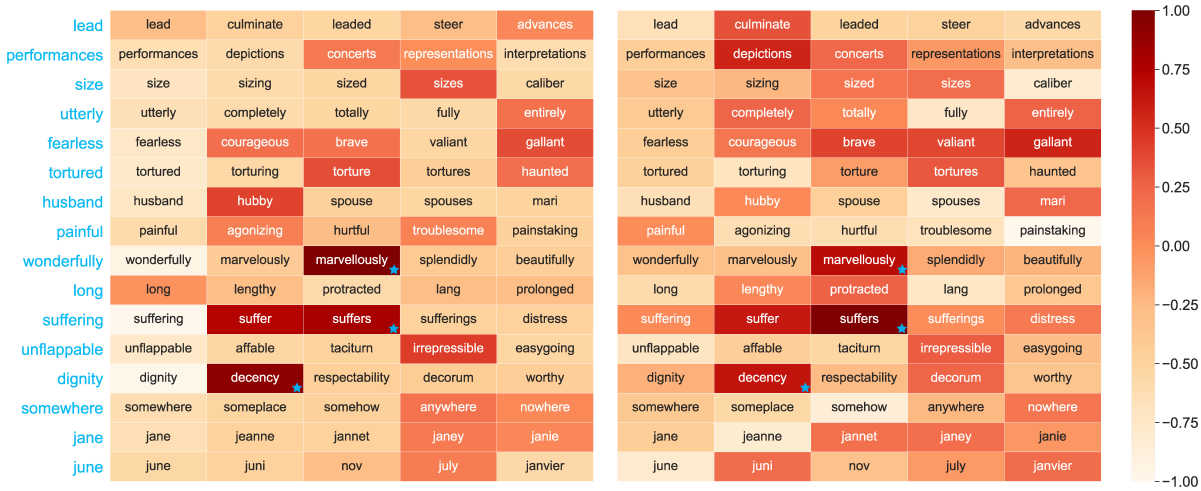
Figure 5: The attack success rate (%) ↑ and perturbation rate (%) ↓ of TextHacker on BERT using IMDB dataset, when varying the neighborhood size δ, population size S or maximum number of local search N.

ples, *i.e.*, positive or negative, and score for the similarity between the benign sample and its adversarial example from 1 (very similar) to 5 (very different). The survey results show that 84.5% of the adversarial examples on TextHacker (*vs*. 79.0% on HLBB and 81.5% on TextHoaxer) are labeled the same as the original samples, and the average similarity score is 1.9 (*vs*. 2.4 on HLBB and 2.1 on TextHoaxer). It demonstrates that the adversarial examples generated by TextHacker are of higher quality and harder to be detected by humans than that of HLBB and TextHoaxer.

D More Visualizations of Weight Table

Here we present more case studies as the extension of Section 4.6 in Figure 6, 7, 8, and the adversarial examples generated by various hard-label attacks in Table 9, 10, 11. These visualizations further verify the consistency between the weight table and the word importance table, proving the effectiveness of the learned weight table in TextHacker.

Original Text.	Label: Positive	Both lead performances are oscar size quaid is utterly fearless as the tortured husband living a painful lie, and moore wonderfully underplays the long suffering heroine with an unflappable 50s dignity somewhere between jane wyman and june cleaver.
Adversarial Text.	Label: Negative	Both lead performances are oscar size quaid is utterly fearless as the tortured husband living a painful lie, and moore marvellously underplays the long suffers heroine with an unflappable 50s decency somewhere between jane wyman and june cleaver.



Weight Table Word Importance Table

Figure 6: Visualization of the weight table in TextHacker and the word importance table from the victim model, representing the word importance of nouns, verbs, adjectives, adverbs, and their candidate words in the original text as shown in Table 9. The original words are highlighted in Cyan, with each row representing the candidate words. The substituted words are highlighted in **Red** with marker ⋆. A darker color indicates a more important word.

Attack	Original Text & Adversarial Example	Prediction
Original Text	Both lead performances are oscar size quaid is utterly fearless as the tortured husband living a painful lie, and moore wonderfully underplays the long suffering heroine with an unflappable 50s dignity somewhere between jane wyman and june cleaver.	Positive
HLBB	Both lead (**leaded**) performances are oscar size quaid is utterly fearless (**brave**) as the tortured (**tortures**) husband (**hubby**) living a painful (**agonizing**) lie, and moore wonderfully underplays the long suffering (**suffer**) heroine (**smack**) with an unflappable 50s dignity (**decency**) somewhere between jane wyman and june cleaver.	**Negative**
TextHoaxer	Both lead performances are oscar size quaid is utterly fearless as the tortured (**tortures**) husband (**hubby**) living a painful (**agonizing**) lie, and moore wonderfully underplays the long suffering (**suffers**) heroine (**smack**) with an unflappable (**easygoing**) 50s dignity somewhere (**nowhere**) between jane wyman and june cleaver.	**Negative**
TextHacker	Both lead performances are oscar size quaid is utterly fearless as the tortured husband living a painful lie, and moore wonderfully (**marvellously**) underplays the long suffering (**suffers**) heroine with an unflappable 50s dignity (**decency**) somewhere between jane wyman and june cleaver.	**Negative**

Table 9: The original text from MR dataset and the adversarial example generated by various hard-label attacks (HLBB, TextHoaxer and TextHacker) on BERT. We highlight the words replaced by the attacks in **Red**. The corresponding original words are highlighted in Cyan.

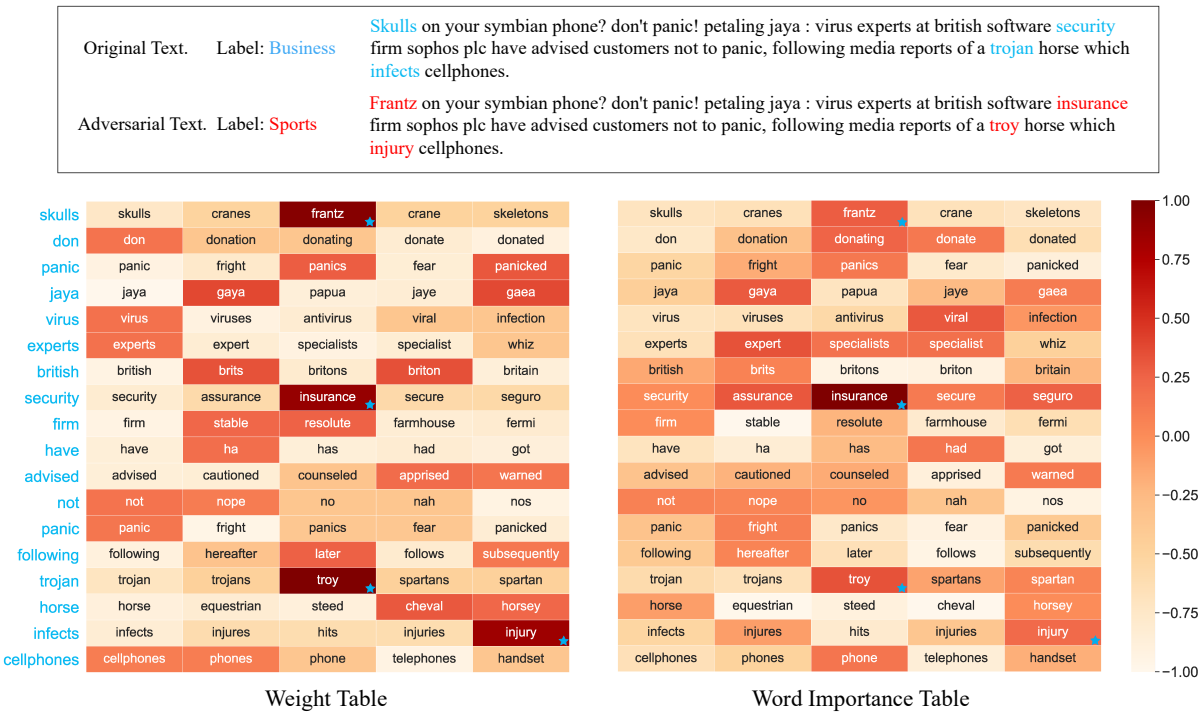
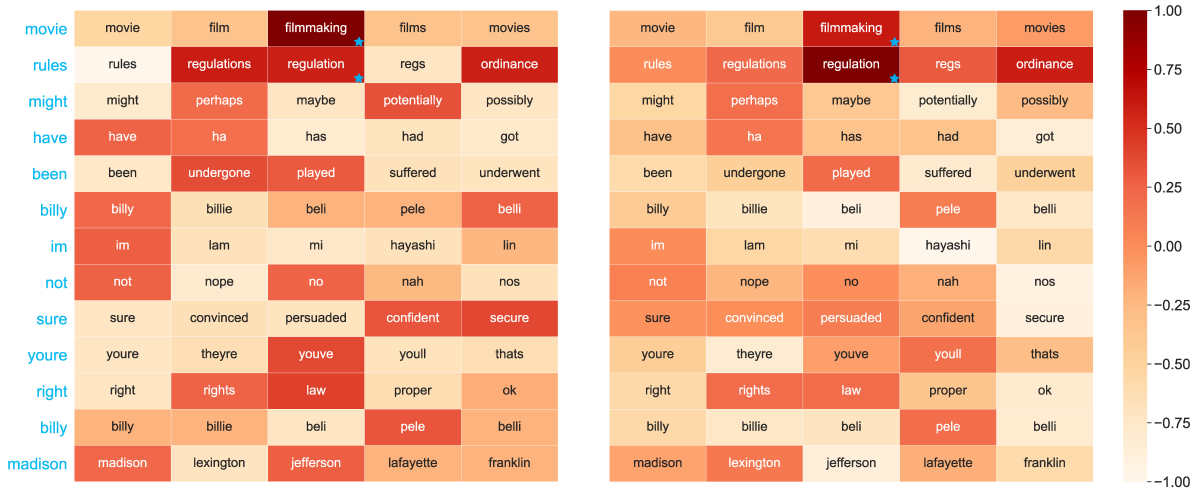


Figure 7: Visualization of the weight table in TextHacker and the word importance table from the victim model, representing the word importance of nouns, verbs, adjectives, adverbs, and their candidate words in the original text as shown in Table 10. The original words are highlighted in Cyan, with each row representing the candidate words. The substituted words are highlighted in **Red** with marker ⋆. A darker color indicates a more important word.

Attack	Original Text & Adversarial Example	Prediction
Original Text	Skulls on your symbian phone? don't panic! petaling jaya : virus experts at british software security firm sophos plc have advised customers not to panic, following media reports of a trojan horse which infects cellphones.	Business
HLBB	Skulls on your symbian phone? don't panic! petaling jaya : virus (**infection**) experts at british software (**sw**) security firm sophos plc have advised customers not to panic, following media reports of a trojan (**spartans**) horse which infects (**injury**) cellphones (**telephones**).	**Sports**
TextHoaxer	Skulls on your symbian phone? don't panic! petaling jaya (**gaya**) : virus experts at british software (**sw**) security (**insurance**) firm (**resolute**) sophos plc have advised customers not to panic, following media reports of a trojan (**spartans**) horse which infects cellphones.	**Sports**
TextHacker	Skulls (**Frantz**) on your symbian phone? don't panic! petaling jaya : virus experts at british software security (**insurance**) firm sophos plc have advised customers not to panic, following media reports of a trojan (**troy**) horse which infects (**injury**) cellphones.	**Sports**

Table 10: The original text from AG's News dataset and the adversarial example generated by various hard-label attacks (HLBB, TextHoaxer and TextHacker) on BERT. We highlight the words replaced by the attacks in **Red**. The corresponding original words are highlighted in Cyan.

Original Text.	Label: Entertainment & Music	What movie is the saying odoyle rules in ?? I think it might have been billy madison but I'm not sure. Yes you're right Billy Madison.
Adversarial Text.	Label: Education & Reference	What filmmaking is the saying odoyle regulation in ?? I think it might have been billy madison but I'm not sure. Yes you're right Billy Madison.



Weight Table Word Importance Table

Figure 8: Visualization of the weight table in TextHacker and the word importance table from the victim model, representing the word importance of nouns, verbs, adjectives, adverbs, and their candidate words in the original text as shown in Table 11. The original words are highlighted in Cyan, with each row representing the candidate words. The substituted words are highlighted in **Red** with marker ⋆. A darker color indicates a more important word.

Attack	Original Text & Adversarial Example	Prediction
Original Text	What movie is the saying odoyle rules in ?? I think it might have been billy madison but I'm not sure. Yes you're right Billy Madison.	Entertainment & Music
HLBB	What movie (**filmmaking**) is the saying (**proverb**) odoyle rules in ?? I think it might have been billy madison but I'm not (**no**) sure (**secure**). Yes you're right Billy Madison.	**Education & Reference**
TextHoaxer	What movie (**filmmaking**) is the saying (**proverb**) odoyle rules in ?? I think it might (**perhaps**) have (**ha**) been (**undergone**) billy madison but I'm not sure. Yes you're right Billy Madison.	**Education & Reference**
TextHacker	What movie (**filmmaking**) is the saying odoyle rules (**regulation**) in ?? I think it might have been billy madison but I'm not sure. Yes you're right Billy Madison.	**Education & Reference**

Table 11: The original text from Yahoo! Answers dataset and the adversarial example generated by various hard-label attacks (HLBB, TextHoaxer and TextHacker) on BERT. We highlight the words replaced by the attacks in **Red**. The corresponding original words are highlighted in Cyan.