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

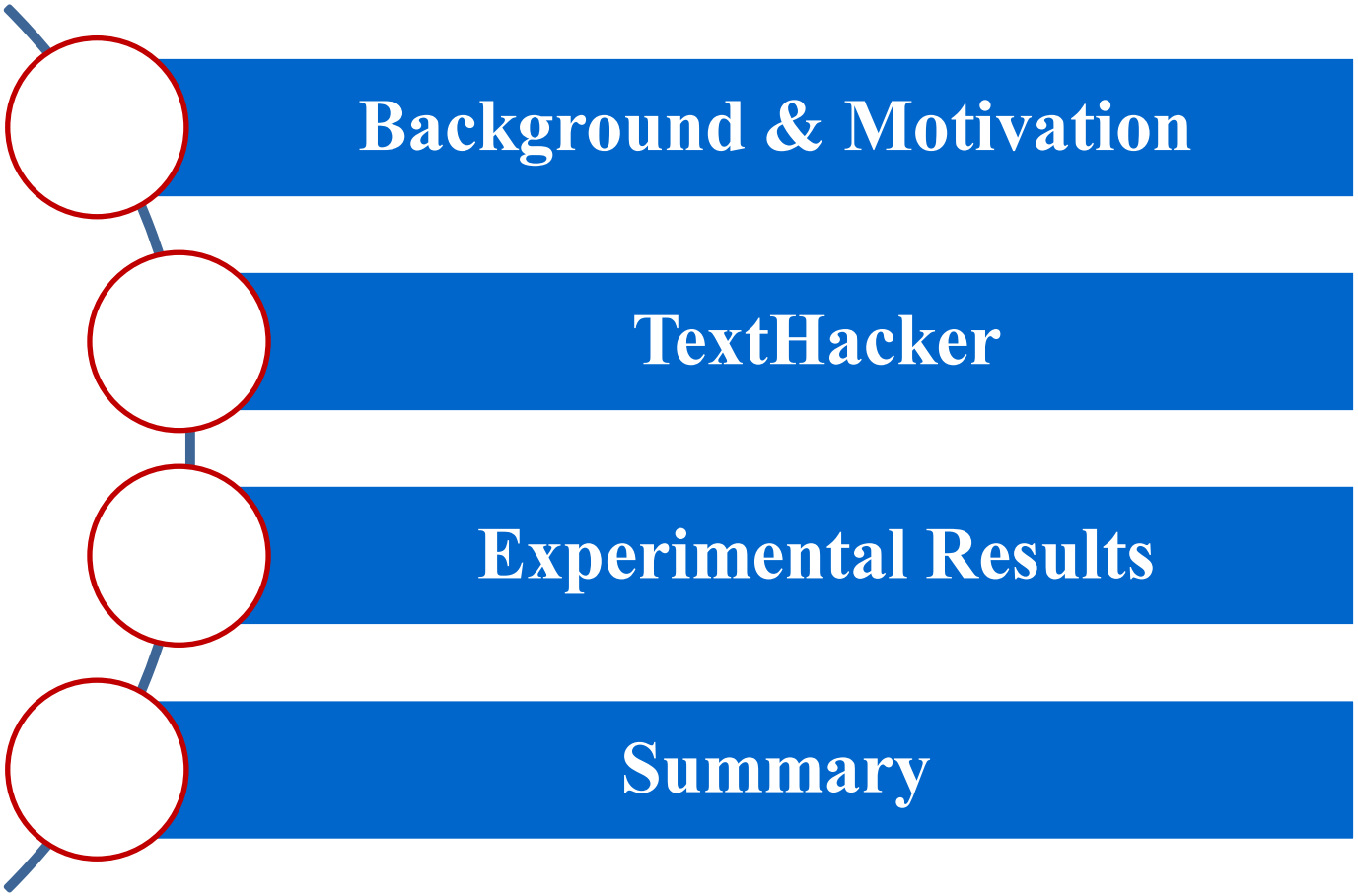
**Zhen Yu**[1], Xiaosen Wang[1,2], Wanxiang Che[3],Kun He[1]

[1] School of Computer Science, Huazhong University of Science and Technology
[2] Huawei Singular Security Lab
[3] Research Center for SCIR, Harbin Institute of Technology
**Contact**: baiding15@hust.edu.cn

December, 2022

# Catalogue

EMNLP 2022



# Textual Adversarial Example

EMNLP 2022

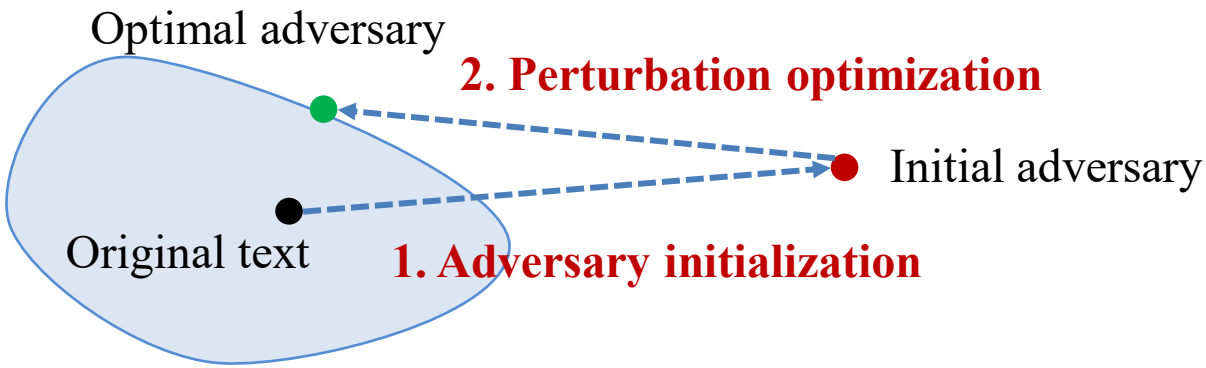
DNNs in NLP tasks are known to be vulnerable to adversarial examples, in which **imperceptible modification** on the correctly classified samples could **mislead the model**.

**Hard-label Attack**: a kind of **Black-Box Attack**. Attacker can only access the model **hard prediction label**, which is more applicable in **real-world** scenarios but also more challenging.

**Background**: 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.

# Related works

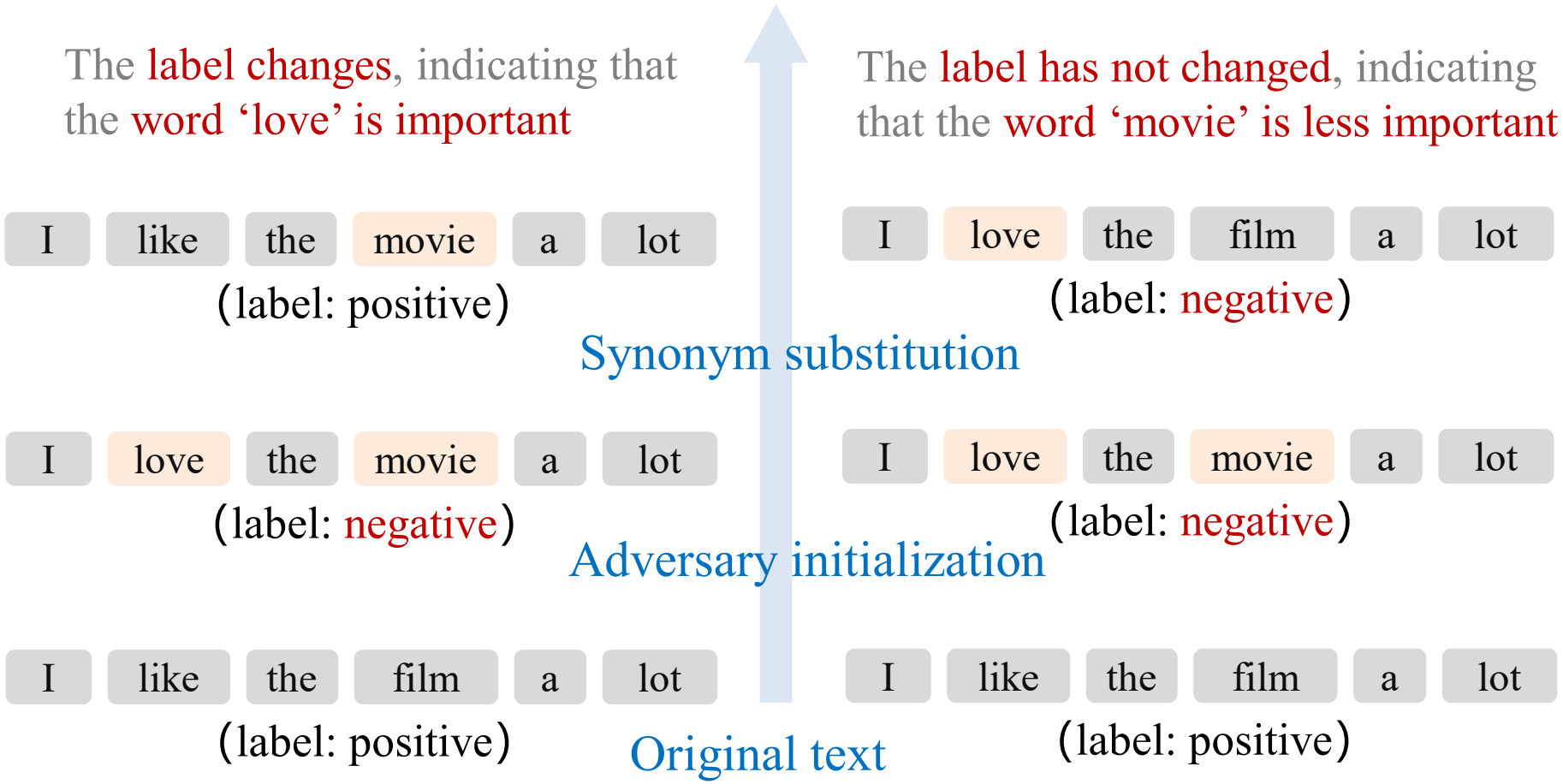
**Existing Hard-label Attacks** usually contain two stages, namely **adversary initialization** and **perturbation optimization**.




- **HLBB** [Maheshwary et al., 2021]: Adopts a genetic algorithm to search for the optimal adversarial example at the perturbation optimization stage.
- **TextHoaxer** [Ye et al., 2022]: Optimizes the perturbation matrix in the continuous embedding space to maximize the semantic similarity and minimize the number of perturbed words.

# Motivation

We **learn the importance of each word** based on the changes of label which can **guide us to minimize the word perturbation**



# TextHacker: Symbols and Definitions

EMNLP 2022

- **Candidate set** $C(w_i)$ : For each word $w_i \in x$, we construct the candidate set $C(w_i)$ containing its top $m$ nearest synonyms according to the distance in the embedding space.
- **Weight table** $W$ : A matrix with the shape of $(n, m+1)$ with all 0s, in which each item $W_{i,j}$ represents the word importance of $\widehat{w}_{i,j} \in C(w_i)$ and $W_{i,:} = \sum_{j=0}^{n} W_{i,j}$ denotes the position importance of word $w_i \in x$.
- **$\delta$-neighborhood $N_\delta(x)$** : A set of texts with at most δ different words from the sample x:

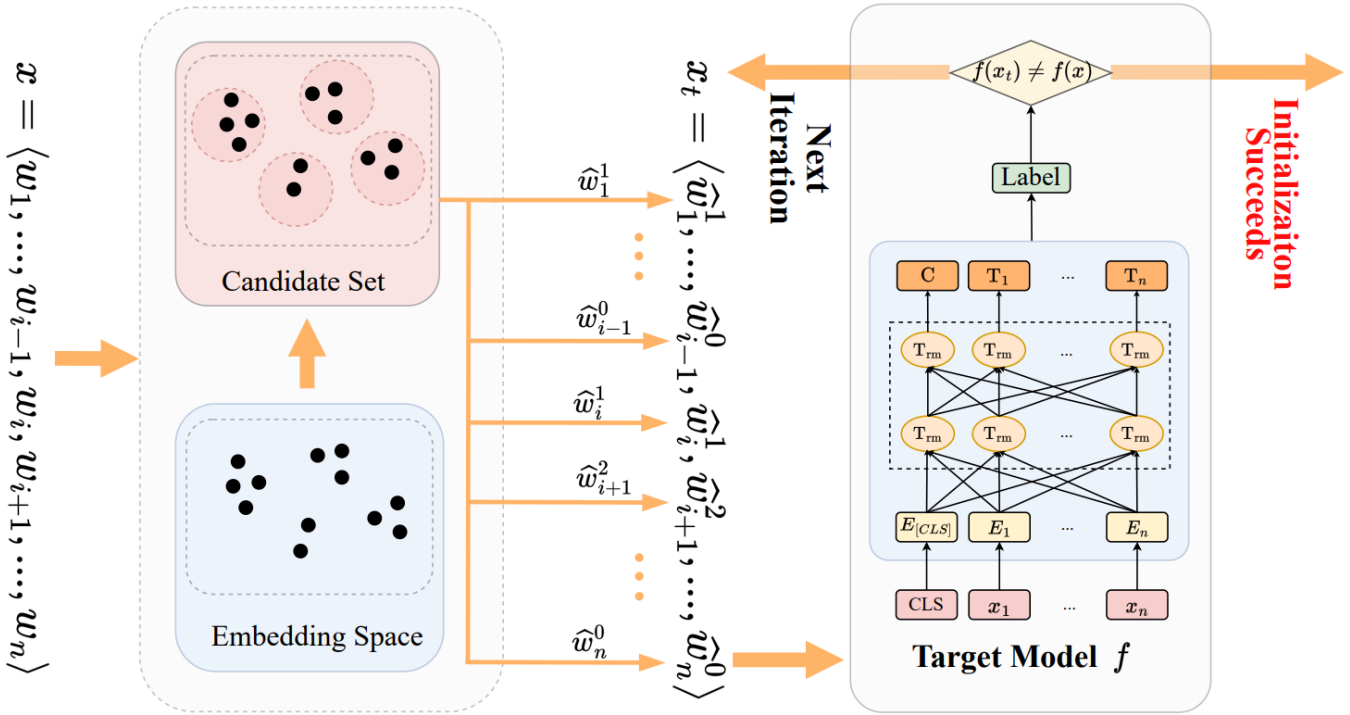
$$N_\delta(x) = \left\{ x^k \mid \sum_{i=1}^{n} \mathbb{1}\left(w_i^k \neq w_i\right) \leq \delta \right\}$$

where $w_i^k \in x^k, w_i \in x$ and $\delta$ is the maximum radius of the neighborhood.

# TextHacker: Adversary Initialization

EMNLP 2022

We **randomly substitute each word** with a candidate word to craft a new text until we **find an adversarial example**.



# TextHacker: Perturbation Optimization

EMNLP 2022

We adopt **the hybrid local search algorithm with the weight table**, a population-based algorithm that contains **local search, weight update** and **recombination** operators, to minimize the adversary perturbation.

- **Local search** 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.
- **Weight update** highlights the important words and positions by assigning different reward for each operated word, which helps the local search select more critical positions and synonyms to substitute.
- **Recombination** crafts non-improved solutions by randomly mixing two adversarial examples, which globally changes the text to avoid poor local optima.

# TextHacker: Perturbation Optimization

EMNLP 2022

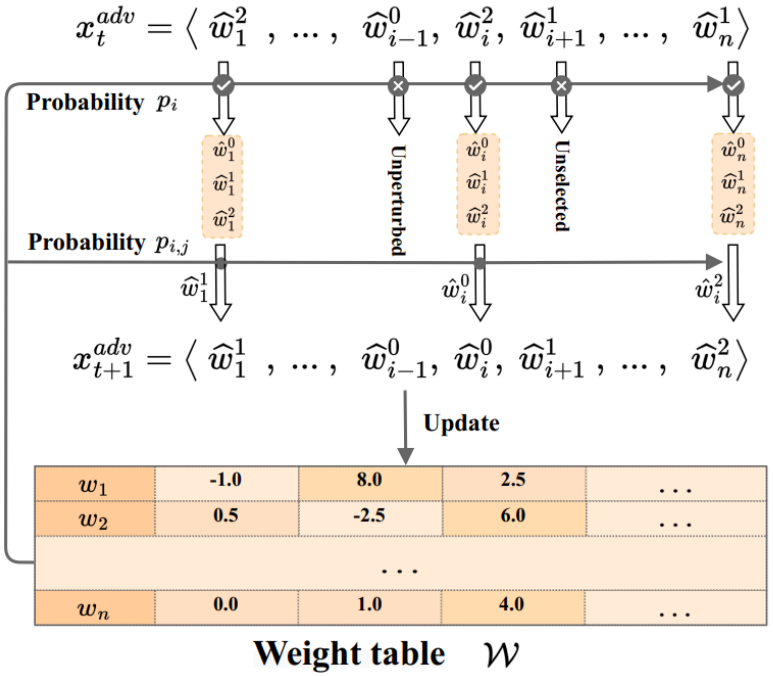
## Local Search:

1. We fist sample several (at most δ) words $\widehat{w_i^{j_t}} \in x_{adv}^t$ with the probability $p_i$ from all the perturbed words in $x_t^{adv}$.

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

2. Then, we substitute each chosen word $\widehat{w_i^{j_t}}$ with the original word $\widehat{w_i^0}$ or with an arbitrary word $\widehat{w_i^{j_{t+1}}} \in C(w_i)$ using the probability $p_{i,j_{t+1}}$ equally to generate a new sample $x_{t+1}^{adv}$.

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



# TextHacker: Perturbation Optimization

EMNLP 2022

**Weight Update** updates the weight table according to the results obtained by local search.

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

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

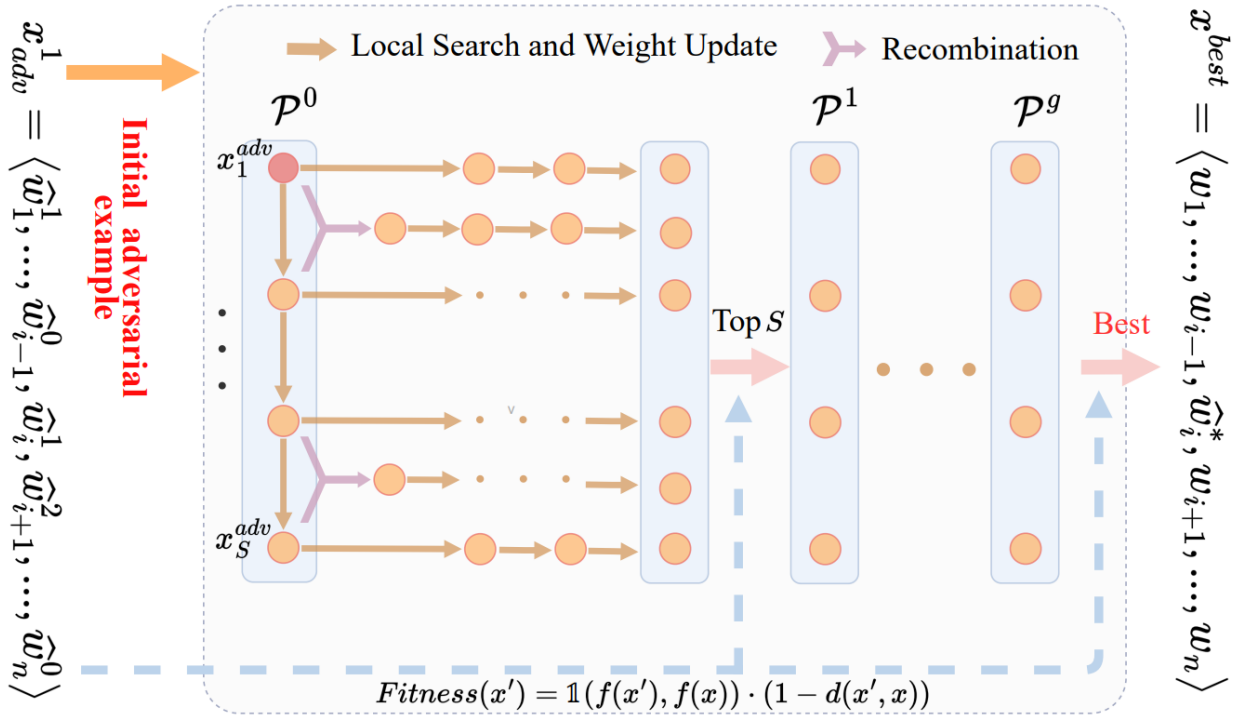
**Rule II**: For each operated position $i$, if $x_{t+1}^{adv}$ is still adversarial, it has little impact on the adversary generation. So we decrease the position weight $W_{i,:}$, and vice versa.

# TextHacker: Perturbation Optimization

EMNLP 2022

**In Summary**:

We first utilize local search to construct an initial population. Subsequently, we iteratively adopt recombination as well as local search to minimize the adversary perturbation, and update the weight table after each local search



# Experimental Settings

EMNLP 2022

- **Dataset**: AG's News, IMDB, MR, Yelp and Yahoo! Answers datasets for text classification. SNLI and MultiNLI dataset for textual entailment.
- **Models**: WordCNN, WordLSTM and BERT for text classification. BERT for textual entailment.
- **Baselines**: Two hard-label attacks, i.e., HLBB and TextHoaxer and two score-based attacks, i.e., GA and PSO for reference.
- **Hyper-parameters**: Neighborhood size $\delta = 5$, reward $r = 1$, population size $S = 4$, maximum number of local search $N = 8$.

# Experimental Results

EMNLP 2022

| 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.

- Better attack performance than existing hard-label attacks.
- Comparable or even better attack performance than the advanced score-based attacks.

| 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.

# Experimental Results

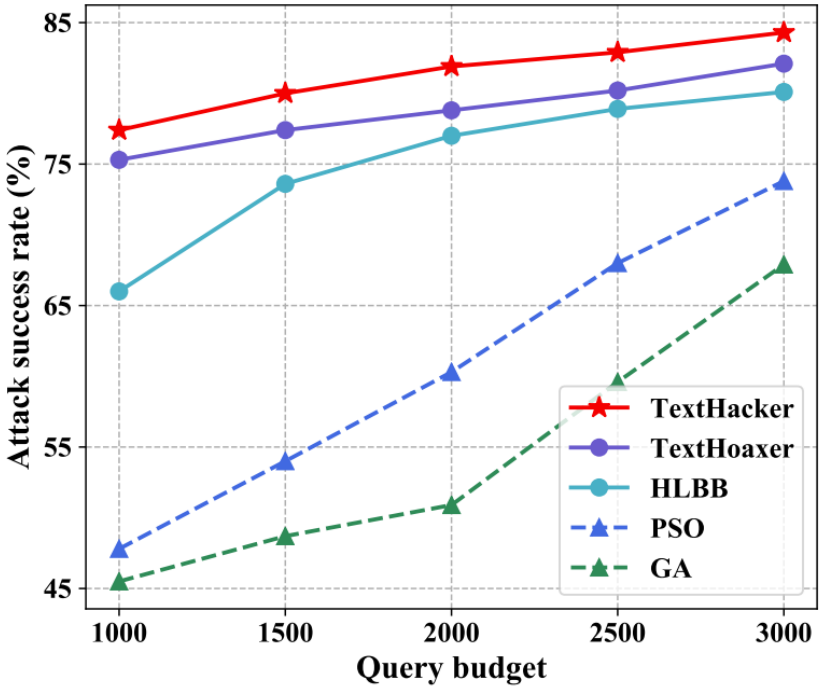



Figure 3: Attack success rate (%) ↑ of various attacks on BERT using IMDB dataset under various query budgets.

TextHacker consistently exhibits better attack performance under various query budgets

# Experimental Results

EMNLP 2022

| 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.

The evaluation on adversary quality demonstrates the high lexicality, semantic similarity and fluency of the generated adversarial examples of TextHacker

# Experimental Results

EMNLP 2022

| 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.

The evaluation on real-world applications demonstrates TextHacker is more practical in real-world scenarios.

# Summary

EMNLP 2022

- Propose a novel text hard-label attack, called TextHacker, which captures the words that have higher impact on the adversarial example via the changes on prediction label to guide the search process at the perturbation optimization stage.
- 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.

# Thanks!

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

[1] School of Computer Science, Huazhong University of Science and Technology
[2] Huawei Singular Security Lab
[3] Research Center for SCIR, Harbin Institute of Technology
**Contact**: baiding15@hust.edu.cn

December, 2022