TRAttack: Text Rewriting Attack Against Text Retrieval

Junshuai Song, Jiangshan Zhang, Jifeng Zhu, Mengyun Tang, Yong Yang
Tencent, China
{jasonjssong,jiangszhang,jifengzhu,mengyuntang,
coolcyang}@tencent.com

Abstract

Text retrieval has been widely-used in many online applications to help users find relevant information from a text collection. In this paper, we study a new attack scenario against text retrieval to evaluate its robustness to adversarial attacks under the black-box setting, in which attackers want their own texts to always get high relevance scores with different users' input queries and thus be retrieved frequently and can receive large amounts of impressions for profits. Considering that most current attack methods only simply follow certain fixed optimization rules, we propose a novel text rewriting attack (TRAttack) method with learning ability from the multi-armed bandit mechanism. Extensive experiments conducted on simulated victim environments demonstrate that TRAttack can yield texts that have higher relevance scores with different given users' queries than those generated by current state-of-the-art attack methods. We also evaluate TRAttack on Tencent Cloud's and Baidu Cloud's commercially-available text retrieval APIs, and the rewritten adversarial texts successfully get high relevance scores with different user queries, which shows the practical potential of our method and the risk of text retrieval systems.

1 Introduction

Text retrieval is a popular and important technology for solving information explosion. In many commercial systems, such as Baidu Knows[1], Answer[2] and StackExchange[3], text retrieval is the key to find relevant content and help search engines to return the information that users want (Trotman et al., 2014). With the development of deep neural networks, many deep learning-based models (Kalchbrenner et al., 2014; Devlin et al., 2018; Sun et al., 2019) are proposed for measuring text relevance. Though the quality of retrieval results is greatly improved, these deep learning-based models (Li et al., 2019a; Song et al., 2020) also bring unexpected serious risks to the text retrieval systems due to their vulnerability.

User Input Queries	Ori.	Adv.
怎么锻炼逻辑思维能力? How to exercise logical thinking skills?	✓	✓
想提升理解能力和逻辑能力? Want to enhance comprehension and logical skills?	×	✓
填字游戏能提高逻辑思维吗? Can crosswords enhance logical thinking?	×	✓
Ori.: 怎么锻炼逻辑思维能力, 让自己更加有效率的学习和工作? How to exercise logical thinking skills to study and work more efficiently? Adv.: 怎么锻炼逻辑思维能力, 让自己方为有智用的自课和工作?		

Table 1: The retrieval results of three different user input queries on an original text (Org. for short) and the adversarial rewritten text (Adv. for short), in which the green words in Org. are replaced with the red words for adversarial goals. '✓' represents that the text is retrieved by the corresponding query.

In this paper, we study a new attack problem in the text retrieval area, in which texts are ranked based on their relevance scores with different user queries. Previous researchers have studied adversarial attacks on retrieval systems (Li et al., 2019a, 2021a). However, the attack goal is to completely subvert the top-k retrieval results of a given single query, with which attackers can deceive the target information retrieval system into retrieving irrelevant content for evading the censorship of professional monitors. Different from the above attack problem, here we focus on a new attack goal

[1] https://zhidao.baidu.com/
[2] https://www.answers.com/
[3] https://stackexchange.com/

Proceedings of the 7th Workshop on Representation Learning for NLP, pages 191 - 203
May 26, 2022 ©2022 Association for Computational Linguistics

where attackers aim to find adversarial texts that can get high relevance scores with many different user queries at the same, and thus there is a high probability for their texts to be retrieved and receive large amounts of impressions (Li et al., 2019b).

This new text retrieval attack problem is realistic as attackers always want more impressions and get more profits than normal users. Table 1 illustrates an attack example, in which the adversarial text is successfully retrieved by all three queries while the original text can be retrieved by one of them only. Attackers can obtain much more impressions and thus get more profits from the text retrieval platform. To verify how serious this form of attack is and facilitate the development of the corresponding countermeasures, we emphasize that it is crucial to develop practical attack methods that can find adversarial texts against existing text retrieval systems.

Query-based adversarial example generation frameworks (Morris et al., 2020; Zeng et al., 2021) could be a good solution for solving the above attack problem under the black-box setting. These methods continuously interact with the victim environment and then iteratively update the generated adversarial examples by received reward signals. However, most of them only simply follow certain fixed optimization rules (Li et al., 2018; Alzantot et al., 2018; Zang et al., 2019) to generate adversarial examples. In other words, they only optimize the adversarial results instead of the attack policies, which greatly limits their attack performance.

For launching attacks more effectively, we propose a novel text rewriting attack (TRAttack) method that can optimize attack policies and examples at the same time by learning from the historical attack knowledge. TRAttack follows the word replacement framework so that it can preserve semantic consistency and language fluency of adversarial examples well. For learning from attack knowledge, we choose reinforcement learning (Sutton and Barto, 2018) to carefully balance the exploration and exploitation in the learning process due to the small number of training samples and expensive interactive costs with the victim environments. Specifically, we choose the well-known multi-armed bandit (MAB) (Kuleshov and Precup, 2014; Lattimore and Szepesvári, 2020; Li et al., 2021b) method. With MAB, the substitutes of each word are viewed as arms to be selected and TRAttack iteratively updates their sampling weights by evaluating the expected adversarial rewards in following iterations for better attack performance. Our main contributions are summarized as follows:

- We discuss a new possible attack threat in text retrieval and formulate the corresponding attack problem to study its robustness to adversarial attacks.
- We develop a novel reinforcement learning-based query-efficient text rewriting attack (TRAttack) method that can achieve high attack performance against text retrieval under the black-box setting.
- We compare TRAttack with existing popular query-based methods and TRAttack achieves much better attack performance. We also successfully attack commercial APIs provided by Tencent Cloud[4] and Baidu Cloud[5], which shows the potential risks of text retrieval systems as APIs could be used in real online applications.

2 Related Work

Language Modeling With the development of deep learning-based natural language processing (Devlin et al., 2018; Cui et al., 2020; Xiao et al., 2020), the quality of text retrieval has been greatly improved in recent years. RNN (Chung et al., 2014; Lipton et al., 2015) is a typical way to encode sequential text information, while convolutional neural networks (CNN) (Liu et al., 2018) and attention-based modeling methods (Vaswani et al., 2017; Zhou et al., 2018) are also used to extract high-dimensional representations for texts. BERT (Devlin et al., 2018; Cui et al., 2020) is a transformer-based method that is bidirectionally trained and has a deeper sense of language context, presenting state-of-the-art results in a wide variety of NLP tasks. Further, many variants based on BERT are proposed and achieve better performance, such as SpanBERT (Joshi et al., 2020), ERNIE (Sun et al., 2019; Xiao et al., 2020), etc. These language modeling methods can be adopted in text retrieval and have boosted the quality of retrieval results (Sakata et al., 2019).

[4]https://cloud.tencent.com/
[5]https://ai.baidu.com/

Adversarial Methods in NLP We consider the most realistic and challenging black-box attack scenario, where attackers have no prior knowledge of the victim model. They can only interact with the victim model to get useful information and optimize their attacks (Zang et al., 2020; Zeng et al., 2021; Morris et al., 2020). Li et al. (2018) follow the idea of greedy word replacement and propose TextBugger. TextFooler (Jin et al., 2020) and PWWS (Ren et al., 2019) are similar to TextBugger, but both of them make stricter restrictions on every single modification for generating plausible and semantically similar adversarial examples. Alzantot et al. (2018) develop Genetic via genetic algorithms. Zang et al. (2019) further propose PSO based on a particle swarm optimization-based search algorithm to generate adversarial examples. BERT-Attack (Li et al., 2020) and BAE (Garg and Ramakrishnan, 2020) use pre-trained masked language models exemplified by BERT to achieve adversarial goals while the generated examples are fluent and semantically preserved.

3 Text Rewriting Attack

In this section, we first formally define the new attack problem against text retrieval under the black-box setting and then introduce the details of our proposed text rewriting attack method.

3.1 Problem Definition

For a query input q, retrieval systems return a list of texts: $X_q = \{x_1, x_2, ..., x_k \mid f(q, x_i) \leq f(q, x_j), s.t.\ i \leq j\}$ ordered by their relevance scores (or similarities) with q where $f(\cdot)$ is the relevance function and k is the size of X_q. As shown in Table 1, attackers' goal is to generate adversarial texts that have 'abnormally' high relevance to many given user queries meanwhile, so that there is a high probability for their texts to be retrieved and thus they can receive large amounts of impressions.

For a text x, we use n_x to represent the number of impressions that it receives in a period of time. Formally, it can be calculated as:

$$n_x = \sum_q s(q, x) \tag{1}$$

where $s(q, x)$ represents whether the text x is retrieved by the query q. Then, the attackers' goal is to find a text x_{adv} that can get $n_{x_{adv}}$ as high as possible.

To make n_x computable in our experiments, we set $s(q, x) > 0$ when x belongs to the top-k relevance texts of the query q, otherwise we have $s(q, x) = 0$. Considering that higher ranking orders usually represent larger probabilities to be exposed to users, we further specifically define $s(q, x) = (k - r + 1)/k$ to assign higher values for texts that have higher ranking orders $r \in [1, k]$ under a query q. We have $s(q, x) \in [0, 1]$. Besides, we define Q_x as the query set that a retrieval system receives in a period of time where the queries and x are on the same topic. Then the objective function can be approximately written as:

$$\arg\max_{x_{adv}} \sum_q^{Q_t} s(q, x_{adv}) \tag{2}$$

where the adversarial goal is to find the text x_{adv} that can always receive high ranking orders under given relevant queries in Q_x and thus maximize $n_{x_{adv}} = \sum_q^{Q_t} s(q, x_{adv})$. Note, retrieval systems calculate relevance scores for ranking different query-text pairs, but these scores are not available to attackers. They can only optimize their attack goals with statistical signals. In our experiments, we adopt the above approximated $n_{x_{adv}}$ in Equation 2 as the adversarial goal under the black-box setting, and also use it to guide the optimization of adversarial attacks.

3.2 Text Rewriting Algorithm

Text rewriting can be implemented by directly generating adversarial texts from scratch (Lipton et al., 2015; Zang et al., 2020) or replacing partial words in the original text only (Li et al., 2020). Since the perturbation budget in the second word replacement framework can be easily bounded to preserve the fluencies and semantics of adversarial texts (Li et al., 2020; Garg and Ramakrishnan, 2020), we also adopt it in TRAttack. The key difference is that there is a particularly-designed memory in TRAttack for caching historical' attack knowledge. Specifically, the memory learns effective word replacement policies that can greatly boost the attack performance. We carefully launch the solution based on MAB, which achieves a good balance between exploration and exploitation as the attack goes on. How to sample from and update the memory are two important questions. In the following, we first introduce the core idea and structure of the memory H in TRAttack, and then give the details of the solutions for the above two questions.

Memory Design with MAB With the MAB mechanism, we need to store some specific information for balancing the exploration and exploitation in the learning process. Specifically, we choose the upper confidence bound (UCB) bandit method in TRAttack. Equation 3 illustrate that how UCB chooses actions (arms) based on existing knowledge:

$$a^* = \arg\max_a r(a) + c\sqrt{\frac{\ln m}{N(a)}} \tag{3}$$

where $r(a)$ is the estimated reward of choosing the arm a, $N(a)$ is the number of times that arm a has been selected before and m is the overall number of players done on the current bandit problem. $r(a)$ and $c\sqrt{\frac{\ln m}{N(a)}}$ represent the exploitation part and the exploration part in UCB, respectively. c is a hyper-parameter to control the level of exploration. At the beginning, UCB encourages exploration as $c\sqrt{\frac{\ln m}{N(a)}}$ is relatively large with a small $N(a)$ for each arm. With the learning process, UCB will concentrate on exploitation, selecting the arm with the highest estimated reward.

Memory in TRAttack TRAttack follows the word replacement framework for generating adversarial examples, and we equip TRAttack with MAB in the word replacement process. Specifically, for a word w, the substitutes of it are viewed as arms to be selected in MAB. We design the memory $H_w = [(s_1, r(s_1), N(s_1)), ..., (s_j, r(s_j), N(s_j))]$ for each word w to store specific information about its substitutes (arms), where s_j represents the j-th potential substitute. $r(s_j)$ and $N(s_j)$ are the estimated reward of replacing w with s_j and the number of times that w has been replaced by s_j before. With H_w on every word w, we can conduct the word replacement policy similar to Equation 3. However, it is costly to fully explore the search space as the standard UCB does because there is a large number of potential substitutes for each word in TRAttack.

To optimize the efficiency of convergence, we further make two updates in TRAttack. First, we manually set the maximum number of substitutes for each word to $L = 200$ to reduce the search space and thus speed up the model convergence. Secondly, we use a function $g(m)$ negatively correlated with m to replace the original hyper-parameter c in Equation 3, with which we can actively reduce exploration in the learning process and further accelerate the model convergence. Though the above two updates may lead to sub-optimal results, it is necessary for TRAttack because of the high learning costs against text retrieval systems in practice. Then, we have Equation 4 in TRAttack for selecting word substitutes:

$$\begin{gathered}\arg\max_s r(s) + g(m)\sqrt{\frac{\ln m}{N(s)}} \\ s.t.\ s \in H_w \text{ and } |H_w| \leq L\end{gathered} \tag{4}$$

To make sure that TRAttack will concentrate on exploitation after a few iterations in practice, we generally require that $g(m)\sqrt{\frac{\ln m}{N(s)}}$ tends to 0 with the increase of m even that $N(s)$ of a substitute s is small. In other words, the word substitute selection in TRAttack can gradually totally depend on the substitute reward so that TRAttack can achieve high performance within the expected time frame.

Overall, for each word w, we use a list H_w to store its substitutes with corresponding rewards and accumulated numbers of times that they have been selected. We have $H = \{H_w; w \in W\}$ where W represents the whole word set in a retrieval system. Besides, TRAttack adopts masked language models to generate word substitutes as (Li et al., 2020) for ensuring that the adversarial text is fluent and semantically preserved. As a result, we have an empty H_w for each word w at the beginning. All word substitutes are gradually collected and merged into H with the learning process. More details about the substitute generation and the maintenance of H will be introduced in the following parts.

TRAttack with Memory Algorithm 1 shows the complete text rewriting process of TRAttack for a given text x and it mainly contains 3 steps.

Step 1: Text Expanding (Optional) Considering that there are usually some short texts consisting of a few words only, word replacement may easily result in adversarial examples with obviously different semantics. To overcome the above problem, we propose to expand texts first and then replace the words that are newly added only for well-preserving the text semantic. To achieve this goal, we choose existing famous pre-trained language models (Radford et al., 2019; Zhang et al.,

2020) to expand the original text directly. As language models may generate long texts, we manually stop the text expanding process when meeting the first question mark or full stop.

In such a way, x_{adv} could be viewed as the concatenation of x and an additional expanded trigger text x_t and we have $x_{adv} = \text{concat}(x, x_t)$. Then, our attack goal can be formulated as replacing the words in x_t to improve the relevance scores between x_{adv} and different users' queries. In practice, attackers can even manually expand x instead of adopting language models and thus this step is optional in TRAttack. TRAttack can also directly conduct attacks base on x as most existing methods (Zeng et al., 2021) without text expanding.

Step 2: Word Replacement with Memory For an initialized adversarial text $x_{adv} = \text{concat}(x, x_t)$, we first decide the word replacement order in x_t, and then choose specific word substitutes with the help of the memory H for generating effective adversarial examples.

For the word importance, there have been many solutions for estimating it (Li et al., 2020; Garg and Ramakrishnan, 2020). Here we calculate the word importance of each w in a text x_t by deleting it from x_{adv} and computing the average decrease in the probability of predicting the correct relevance label y with the corresponding queries in Q_x. Then we sort the words in x_t by their importance and get $I = [w_1, w_2, ..., w_{|x_t|}]$ for further word replacement.

For a selected word w to be replaced, we then need to decide the word substitute set S_w for it and conduct the word replacement operation for better attack performance. Following the idea in (Li et al., 2020), we generate word substitutes for a word w by masking it in x_{adv} and feeding the masked x_{adv} into a well-trained masked language model, in which the genuine nature of the masked language model makes sure that the texts with the generated substitutes are relatively fluent and also preserve most semantic information. Each time, we use the top-M predictions from a masked language model to initialize S_w first, and then update S_w with learned H_w for better word replacement choices. On the one hand, for the substitutes that are new and do not appear in H_w before, we use S_w^* to represent them and make sure all the substitutes in S_w^* are selected by default, which helps us to continuously enrich the candidate substitutes of different words. On the other hand, we select

Algorithm 1 Text Rewriting Attack

Input: Text x, query set Q_x, memory $H = \{H_w; w \in W\}$, number of substitutes M, number of memory size L
Output: Adversarial text x_{adv}

```
Expand x and get the initialized x_adv ← concat(x, x_t)
Sort the words in x_t by their estimated importance and get I ← [w_1, w_2, ..., w_|x_t|]
for i ← 1 to |x_t| do
    Generate the top-M substitutes for w_i using masked language models and use them to initialize S_wi
    S*_wi ← S_wi \ (S_wi ∩ H_wi)
    Select M − |S*_w| words from H_wi as S**_wi according to Equation 4
    S_wi ← S*_wi ∪ S**_wi
    for j ← 1 to |S_wi| do
        Get x'_adv by replacing w_i with s_j
        Calculate the reward r'(s_j)
        if r'(s_j) > 0 then
            x_adv ← x'_adv
    Update H_wi
return Adversarial text x_adv
```

the other $M - |S_w^*|$ substitutes from the learned memory H_w for the current word w and get S_w^{**}. Finally, we reconstruct $S_w \leftarrow S_w^* \cup S_w^{**}$.

The selection of substitutes from H_w is based on Equation 4. For $r(s)$, we define it based on the attack performance improvement between x'_{adv} and x_{adv} where x'_{adv} is obtained by replacing w with s in x_{adv}. Specifically, we set $r(s) = (n_{x'_{adv}} - n_{x_{adv}})/|Q_x|$, where $1/|Q_x|$ is used for normalization, and we have $r(s) \in [-1, 1]$. With the learning process, a word substitute s with better historical attack performance will have a larger $r(s)$ and thus have a larger chance to be selected in the future, which can boost the attack performance of TRAttack. For $g(m)$, we define $g(m) = \frac{c}{m}$ and $c = 50$ is a constant. In this setting, $g(m)\sqrt{\frac{\ln m}{N(w)}}$ tends to 0 with the increase of m and thus we can successfully actively reduce exploration for achieving high attack performance within limited attack attempts and costs.

Step 3: Memory Update For each substitute $s \in S_w$ with the newly calculated reward $r'(s)$ in the current iteration, we update H_w following the below rules. If s is new to H_w, we directly merge it into H_w. If s already appears in H_w, we use the following Equation 5 to update $r(s)$ of s in H_w:

$$r(s) = \frac{r(s) * N(s) + r'(s)}{N(s) + 1} \quad (5)$$

And then we set $N(s) \leftarrow N(s) + 1$. Besides, if the size of H_w exceeds L, we additionally remove the substitutes with relatively low $r(\cdot)$ in H_w and make sure that $|H_w|$ does not exceed L.

4 Experiments

4.1 Experimental Settings

Dataset LCQMC (Liu et al., 2018) is a large-scale Chinese question matching corpus collected from Baidu Knows. BQ-Corpus (Chen et al., 2018) contains question pairs from online bank custom service logs. We use these 2 publicly-available datasets for text retrieval in our experiments. Overall, there are 256433 and 35395 different texts in LCQMC and BQ-Corpus, respectively.

Evaluation Metric We adopt 4 metrics for evaluating different attack methods comprehensively. For the attack performance, we define $R(x_{adv}) = n_{x_{adv}}/|Q_x|$ to represent the attack performance of a generated adversarial text x_{adv} where $1/|Q_x|$ is used for normalization. For the quality of adversarial examples, we adopt the common metric perplexity (PPL) (Li et al., 2020) as (Zang et al., 2019), and use the cosine similarity between text embeddings as an approximation for the semantic consistency (Jin et al., 2020). As only x_t in x_{adv} is modified, we test PPL and semantic consistency on it by default. Besides, we report the number of interactions of each method, which is another important metric for evaluating the attack costs.

Text Retrieval Systems Given a user query q, the text retrieval system computes relevance between q and existing texts in the system and then returns the most relevant texts to the user. Due to a large number of the corpus in real-world systems, there are usually two stages for text retrieval: candidate generation and ranking (Yang et al., 2019). In our experiments, we simulate different retrieval systems. Specifically, we adopt the popular BM25 (Trotman et al., 2014) for selecting 100 texts as candidates each time and then use different ranking models to rank them by calculating their relevance scores with different given queries. We choose 4 different representative language models as the ranking model, including LSTM, CNN, BERT (Cui et al., 2020) and ERNIE-Gram (Xiao et al., 2020). Most of the above ranking models have been introduced in Section 2 and their implementation details can be found in Appendix A.

Overall, we use the 4 text retrieval models and the 2 datasets to construct 8 different simulated text retrieval environments as the testbeds for experimental evaluation. If a generated adversarial text x_{adv} frequently receives large relevance scores with different user queries, it will have high ranking orders in users' retrieval results most of the time, leading to a high $R(x_{adv})$. In order to fully demonstrate the changes of the ranking results of generated adversarial examples, we set $k = 100$ that is the number of candidates by default when calculating $R(\cdot)$.

4.2 Comparison with Query-based Attack Baselines

We compare TRAttack with 4 popular query-based adversarial methods that work well under the black-box setting, including TextBugger (Li et al., 2018), PWWS (Ren et al., 2019), Genetic (Alzantot et al., 2018), PSO (Zang et al., 2019) and BERT-Attack (Li et al., 2020). TextBugger and PWWS are greedy methods, in which they first sort words in given texts by importance and then replace them with carefully selected substitutes for achieving adversarial goals. Genetic and PSO are representative population-based search algorithms. For generating effective adversarial examples, both of them first initialize a text set (the size is set to 20 in our experiments) and then iteratively update them with different evolutionary algorithms. For BERT-Attack, it is also a greedy word replacement method and we replace the masked language model in it with Chinese-BERT-wwm (Cui et al., 2019) for conducting adversarial attacks in Chinese. As for our method TRAttack, we adopt Chinese-BERT-wwm as the masked language model for generating word substitutes as well. For the parameters, we set $M = 36$ and $L = 200$ by default. Besides, for a fair comparison, we adopt the optional text expanding process with CPM (Zhang et al., 2020) in all attack methods.

The comparison results[6] are illustrated in Table 2. In each testbed, we randomly choose 500 texts to generate adversarial examples and calculate the average results. As we can see, TRAt-

[6]We discuss the experimental results on the LCQMC dataset in Section 4.2 and the results on the BQ-Corpus dataset are reported in Appendix due to the page limitation.

Method	Num.	Per.	PPL	Sem.
TextBugger	151	0.8126	1106	0.5117
PWWS	197	0.7115	565	0.6625
Genetic	807	0.5599	432	0.8233
PSO	306	0.4705	432	0.8781
BERT-Attack	137	0.7533	995	0.9129
TRAttack	141	0.8341	1159	0.9015

(a) The simulated text retrieval system with LSTM

Method	Num.	Per.	PPL	Sem.
TextBugger	151	0.6519	917	0.4868
PWWS	197	0.5581	532	0.7162
Genetic	807	0.4423	404	0.8252
PSO	306	0.4089	403	0.8852
BERT-Attack	137	0.6514	795	0.9177
TRAttack	141	0.6772	1104	0.9079

(b) The simulated text retrieval system with CNN

Method	Num.	Per.	PPL	Sem.
TextBugger	151	0.6891	841	0.7062
PWWS	197	0.6369	722	0.7059
Genetic	807	0.5436	512	0.8178
PSO	310	0.5028	362	0.8229
BERT-Attack	137	0.6492	971	0.9078
TRAttack	141	0.6636	1355	0.9014

(c) The simulated text retrieval system with BERT

Method	Num.	Per.	PPL	Sem.
TextBugger	151	0.7506	797	0.6819
PWWS	197	0.7107	475	0.7216
Genetic	807	0.6290	335	0.8414
PSO	307	0.5891	321	0.8361
BERT-Attack	137	0.6831	914	0.9126
TRAttack	141	0.7037	1375	0.9086

(d) The simulated text retrieval system with ERNIE-Gram

Table 2: Attack results on different simulated text retrieval systems on the LCQMC dataset. Num., Per. and Sem. represent the number of interactions, the attack performance $R(\cdot)$ and the semantic consistency, respectively.

tack achieves the best results on the whole. In TextBugger, it defines some 'bug' generation ways for adversarial attacks. Though it achieves high attack performance, most of its generated adversarial texts are less fluent and semantically consistent compared with other methods. PWWS follows the greedy word replacement framework. As the synonym-based word substitute generation method with thesauri like WordNet (Miller, 1995) always provides very limited synonyms for many words, we use the embedding-based word substitute generation method as TextBugger in it for better attack performance. In our experiments, PWWS receives lower Per. than TextBugger while PPL and Sem. are usually better.

Genetic and PSO are population-based methods. For PSO, as the sememe-based word substitute generation method (Zang et al., 2019) also greatly reduces the number of potential word substitutes, we adopt the embedding-based word substitute generation method in it as well. In our experiments, both of these two methods receives relatively low attack performance compared with other methods, which may be due to the fact that they usually need a long period of evolution (large Num.) to achieve satisfying results.

BERT-Attack adopts masked language models to generate adversarial examples and receives relatively high Per. and Sem. in our tests at a low attack cost. TRAttack follows the similar framework with it and can further get an obvious improvement on Per., while PPL and Sem. are slightly worse than BERT-Attack. This is due to that we always choose words that can achieve high attack performance from the learned memory in TRAttack, but these newly selected words may slightly damage the fluency and semantic consistence sometimes. Here, to illustrate the advantages of our method more comprehensively, we conduct additional experiments for TRAttack. Specifically, We test TRAttack by reducing different numbers of words that can be replaced by substitutes in it. The results conducted on the simulated text retrieval system based on LSTM and the LCQMC dataset are reported in Table 3.

Value	Num.	Per.	PPL	Sem.
1	127	0.7923	836	0.9153
2	114	0.7478	691	0.9250
3	101	0.7139	557	0.9281
4	88	0.6674	496	0.9395

Table 3: Attack results with different reduced numbers of words that can be replaced.

As we can see, with a larger reduced number of words that can be replaced, TRAttack gradually receives better PPL and Sem. while Per. becomes smaller. An important experimental result is that TRAttack achieves better performance on all the 4 metrics than BERT-Attack when the reduced number is set to 1, which clearly shows the advantages of TRAttack. Overall, we can say that TRAttack achieves the best performance among all compared methods by optimizing the attack policies (memory) and examples meanwhile. Besides, it is worth mentioning that the learned knowledge in TRAttack is general and can be continuously updated with new attack results, which is the key advantage and foundation for TRAttack to further achieve better attack performance in the future. We also illustrate an adversarial example generated by TRAttack in Table 9 in Appendix, which can successfully receive high relevance scores with 10 different queries.

4.3 Parameter Analysis

Tables 4 and 5 show the test results of TRAttack regarding two different hyper-parameters on the simulated text retrieval system based on LSTM and the LCQMC dataset: the number of substitutes M and the memory size L.

Value	Num.	Per.	PPL	Sem.
6	42	0.7206	868	0.9040
12	74	0.7710	966	0.9080
24	113	0.8117	1053	0.9065
36	141	0.8341	1159	0.9015
48	171	0.8429	1233	0.9038

Table 4: Attack results with different M.

Value	Num.	Per.	PPL	Sem.
50	141	0.8175	1138	0.9110
200	141	0.8341	1159	0.9015
500	141	0.8369	1190	0.9067
2000	141	0.8329	1193	0.9033

Table 5: Attack results with different L.

Intuitively, TRAttack can receive better attack performance with a larger M. As we can see in Table 4, the performance improvement gradually becomes insignificant. For balancing the attack performance and other metrics, $M = 36$ could be a good choice for conducting adversarial attacks in practice. As for L, the attack performance can generally be increased along with it increasing. However, as we actively speed up the convergence of TRAttack by $g(m)$ for achieving good performance within limited attack costs, the word replacement policy with a large memory may not be learned well, thus leading to a worse result. As we can see in Table 5, TRAttack receives a relatively good result with $L = 200$.

4.4 Attack Commercial APIs

We have shown that TRAttack can effectively attack simulated text retrieval systems in Section 4.2. Here, we show that TRAttack can also successfully create adversarial texts on commercial text retrieval APIs provided by Tencent Cloud and Baidu Cloud. Due to the QPS limitation, we randomly test 10 samples for both APIs in our experiments.

API	Num.	Per.	PPL	Sem.
Tencent	1761	0.5570	1014	0.8878
Baidu	1712	0.6619	556	0.8951

Table 6: Attack results of TRAttack on the Tencent Cloud's and Baidu Cloud's APIs.

The results are reported in Table 6, in which we iteratively optimize the generated adversarial texts by 10 iterations in TRAttack for better attack performance. As a result, TRAttack successfully generates effective adversarial examples that can increase Per. from 0.3686 to 0.5570 and from 0.4085 to 0.6619 on the Tencent Cloud's and Baidu Cloud's commercial APIs with only about 2000 times of interactions, respectively. Tables 10 and 11 in Appendix show specific attack cases of TRAttack on the commercial APIs. The experiments in this part are conducted as of November 2021.

5 Conclusion

In this paper, we discuss a new realistic attack problem against text retrieval. We follow the word replacement framework and propose TRAttack. Extensive experiments show that benefiting from the the learning ability of MAB, TRAttack achieves better performance than existing methods. The generated adversarial texts by TRAttack can successfully mislead both offline text retrieval models and online commercial APIs, which demonstrates the potential risks of real-world text retrieval systems.

6 Broader Ethical Impact

We explore the potential security issues of text retrieval systems in this paper and propose TRAttack that is experimentally verified to be effective to many text retrieval models. Hope that our approach and discussions could inspire more explorations and designs of advanced defense methods and security policies.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*.

Jing Chen, Qingcai Chen, Xin Liu, Haijun Yang, Daohe Lu, and Buzhou Tang. 2018. The bq corpus: A large-scale domain-specific chinese corpus for sentence semantic equivalence identification. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 4946–4951.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting pre-trained models for Chinese natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 657–668, Online. Association for Computational Linguistics.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. 2019. Pre-training with whole word masking for chinese bert. *arXiv preprint arXiv:1906.08101*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970*.

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 *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Volodymyr Kuleshov and Doina Precup. 2014. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*.

Tor Lattimore and Csaba Szepesvári. 2020. *Bandit algorithms*. Cambridge University Press.

Jie Li, Rongrong Ji, Hong Liu, Xiaopeng Hong, Yue Gao, and Qi Tian. 2019a. Universal perturbation attack against image retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4899–4908.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984*.

Xiaodan Li, Jinfeng Li, Yuefeng Chen, Shaokai Ye, Yuan He, Shuhui Wang, Hang Su, and Hui Xue. 2021a. Qair: Practical query-efficient black-box attacks for image retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3330–3339.

Zhao Li, Junshuai Song, Shichang Hu, Shasha Ruan, Long Zhang, Zehong Hu, and Jun Gao. 2019b. Fair: Fraud aware impression regulation system in large-scale real-time e-commerce search platform. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1898–1903. IEEE.

Zhao Li, Junshuai Song, Zehong Hu, Zhen Wang, and Jun Gao. 2021b. Constrained dual-level bandit for personalized impression regulation in online ranking systems. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(2):1–23.

Zachary C Lipton, John Berkowitz, and Charles Elkan. 2015. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

Xin Liu, Qingcai Chen, Chong Deng, Huajun Zeng, Jing Chen, Dongfang Li, and Buzhou Tang. 2018. Lcqmc: A large-scale chinese question matching corpus. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1952–1962.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 119–126.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1085–1097.

Wataru Sakata, Tomohide Shibata, Ribeka Tanaka, and Sadao Kurohashi. 2019. Faq retrieval using query-question similarity and bert-based query-answer relevance. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1113–1116.

Junshuai Song, Zhao Li, Zehong Hu, Yucheng Wu, Zhenpeng Li, Jian Li, and Jun Gao. 2020. Poisonrec: an adaptive data poisoning framework for attacking black-box recommender systems. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 157–168. IEEE.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. Improvements to bm25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium*, pages 58–65.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Dongling Xiao, Yu-Kun Li, Han Zhang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie-gram: Pre-training with explicitly n-gram masked language modeling for natural language understanding. *arXiv preprint arXiv:2010.12148*.

Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Simple applications of bert for ad hoc document retrieval. *arXiv preprint arXiv:1903.10972*.

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

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2019. Word-level textual adversarial attacking as combinatorial optimization. *arXiv preprint arXiv:1910.12196*.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2021. Openattack: An open-source textual adversarial attack toolkit. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 363–371.

Zhengyan Zhang, Xu Han, Hao Zhou, Pei Ke, Yuxian Gu, Deming Ye, Yujia Qin, Yusheng Su, Haozhe Ji, Jian Guan, Fanchao Qi, Xiaozhi Wang, Yanan Zheng, Jiannan Cao, Guoyang Zeng, Huanqi Cao, Shengqi Chen, Daixuan Li, Zhenbo Sun, Zhiyuan Liu, Minlie Huang, Wentao Han, Jie Tang, Juanzi Li, and Maosong Sun. 2020. Cpm: A large-scale generative chinese pre-trained language model.

Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. 2018. Atrank: An attention-based user behavior modeling framework for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

A Accuracy of Text Retrieval Models

In both LSTM and CNN, We use an embedding layer to encode words firstly. And then we directly use one LSTM layer to extract the text representation in LSTM while using the CNN structure for CNN. With the representations of a given query and a candidate text, the relevance is predicted by feeding concatenated features of both texts into a 2-layer deep neural network (DNN) with Softmax for calculating probabilities. In BERT and ERNIE-Gram, we directly concatenate the two texts inputs, and use BERT and ERNIE-Gram to get the final representation. The results are also predicted by feeding the representation into a 2-layer DNN with Softmax. The size of word embeddings and the DNN in each model is set to be 128. We adopt CrossEntropy as the loss function and use Adam as the optimizer. For the learning rate, we use $\alpha = 2e - 3$ in LSTM and CNN, and $\alpha = 2e - 5$ in BERT and ERNIE-Gram.

Models	LCQMC	BQ-Corpus
LSTM	0.7864	0.6780
CNN	0.7597	0.6671
BERT	0.8888	0.8529
ERNIE-Gram	0.9054	0.8610

Table 7: Accuracy of different models on 2 datasets.

For the model training, we adopt the popular early stopping mechanism for better performance, and Table 7 reports the accuracy of different models on the 2 datasets.

Method	Num.	Per.	PPL	Sem.
TextBugger	145	0.9374	993	0.5286
PWWS	191	0.9296	822	0.5576
Genetic	808	0.7316	681	0.7749
PSO	297	0.6374	353	0.8642
BERT-Attack	127	0.9008	1027	0.9281
TRAttack	132	0.9299	1200	0.9234

(a) The simulated text retrieval system with LSTM

Method	Num.	Per.	PPL	Sem.
TextBugger	145	0.7863	1229	0.5212
PWWS	191	0.7485	749	0.6981
Genetic	808	0.6352	524	0.8233
PSO	298	0.5881	481	0.8842
BERT-Attack	127	0.7969	1110	0.9215
TRAttack	132	0.8383	1452	0.9106

(b) The simulated text retrieval system with CNN

Method	Num.	Per.	PPL	Sem.
TextBugger	145	0.6160	1026	0.5817
PWWS	191	0.5800	921	0.6826
Genetic	808	0.5247	643	0.7900
PSO	298	0.5042	353	0.8591
BERT-Attack	127	0.5916	1703	0.9079
TRAttack	132	0.6004	2294	0.9075

(c) The simulated text retrieval system with BERT

Method	Num.	Per.	PPL	Sem.
TextBugger	145	0.6616	885	0.5553
PWWS	191	0.6271	900	0.6932
Genetic	808	0.5643	641	0.8115
PSO	296	0.5473	554	0.8687
BERT-Attack	127	0.6448	1546	0.9029
TRAttack	132	0.6539	2184	0.9057

(d) The simulated text retrieval system with ERNIE-Gram

Table 8: Attack results on different simulated text retrieval systems on the BQ-Corpus dataset. Num., Per. and Sem. represent the number of interactions, the attack performance $R(\cdot)$ and the semantic consistency, respectively.

User Input Queries	Ori.	Adv.
怎样自己制作文字图片? How to make text pictures?	0.5930 / 0.69	0.9972 / 0.97
谁会自己制作文字图片? Who can make text pictures by yourself?	0.9506 / 0.93	0.9996 / 1.00
哪个网站可以自己制作图片? Which website can we use to make pictures?	0.8650 / 0.35	0.9957 / 0.96
怎样在手机上制作自己的文字图片? How to make my own text pictures on the mobile phone?	0.7229 / 0.26	0.9993 / 0.99
怎么制作自己的网页? How to create my own webpage?	0.1236 / 0.54	0.9962 / 0.94
如何自己制作带音乐、多张图片和文字的电子贺卡? How to make an e-card with music, multiple pictures, and text by myself?	0.9658 / 0.26	0.9996 / 0.99
怎么可以制作自己的网页? How can I make my own webpage?	0.2927 / 0.49	0.9982 / 1.00
火车票图片制作 Train ticket picture making	0.5336 / 0.44	0.9895 / 0.94
自己怎么制作冰淇淋? How to make ice cream by myself?	0.9889 / 0.22	0.9997 / 0.98
读书卡怎样制作? How to make a reading card?	0.9242 / 0.32	0.9999 / 0.98
Ori. : 怎样自己制作文字图片? 有 哪些 软件 可以 帮助我们制 作 文 字 图 片 ? How to make text pictures? Which software can help us make text pictures? Adv.: 怎样自己制作文字图片? 有 那种 软件 支帮 帮助我们制 做 文 本 图 图 ?		

Table 9: A generated adversarial example by TRAttack that successfully receives high $f(\cdot)$ / $s(\cdot)$ under 10 different queries meanwhile on the simulated text retrieval system based on LSTM and the LCQMC dataset.

User Input Queries	Ori.	Adv.
守护甜心第四季什么时候播? When will the fourth season of "Shugo Chara!" be broadcast?	0.5592 / 0.62	0.6417 / 0.64
破产姐妹什么时候播第四季 When will the "Broke Girls" broadcast the fourth season	0.5519 / 0.48	0.6151 / 0.78
活佛济公第四部到底什么时候播 When will the fourth season of "The Legend of Crazy Monk" be broadcast	0.5665 / 0.54	0.6194 / 0.67
爱情回来了什么时候播 When will "Love is Back" be broadcast	0.4966 / 0.07	0.6612 / 0.59
美人制造什么时候播 When will "Cosmetology High" be broadcast	0.4986 / 0.15	0.6627 / 0.69
叶罗丽精灵梦第三季什么时候播? When will the third season of "Yeloli" be broadcast?	0.5067 / 0.28	0.5099 / 0.28
世界上另一个我什么时候播 When will "Another Me in the World" be broadcast	0.5375 / 0.42	0.6524 / 0.72
终极宿舍什么时候播 When will "THE X-DORMITORY" be broadcast	0.4980 / 0.21	0.7200 / 0.78
新少年四大名捕电视剧什么时候播 When will "The Four" be broadcast	0.5294 / 0.44	0.5477 / 0.51
不一样的美男子什么时候播? When will "Special Different Man" be broadcast?	0.4825 / 0.04	0.6579 / 0.76
Ori. : 守护甜心第四季什么时候播? 《老友记》 里有 哪些 经 典 台 词 ? When will the fourth season of "Shugo Chara!" be broadcast? What are the classic lines in "Friends"? Adv.: 守护甜心第四季什么时候播? （小友記秀 里有 谁多 经 经 台 辞 ?		

Table 10: A generated adversarial example by TRAttack that successfully receives high $f(\cdot)$ / $s(\cdot)$ under 10 different queries meanwhile on the Tencent Cloud's commercial API.

User Input Queries	Ori.	Adv.
在家可以做的兼职有什么? What are the part-time jobs that can be done at home?	0.8460 / 0.60	0.9553 / 0.95
在家电脑兼职可以做什么 What are the part-time jobs that can be done on the computer at home	0.7734 / 0.61	0.8474 / 0.88
有没有什么在家就可以做的兼职? Are there any part-time jobs that can be done at home?	0.7795 / 0.51	0.9009 / 0.92
可在家做的兼职? Part-time jobs can be done at home?	0.8129 / 0.55	0.9304 / 0.92
在家兼职的工作有哪些 What are the part-time jobs that can be done at home	0.8449 / 0.65	0.8706 / 0.75
有没有在家能做的兼职? Are there any part-time jobs that can be done at home?	0.7280 / 0.32	0.8558 / 0.83
如何在家做淘宝客服兼职 How to be a part-time Taobao customer service at home	0.6210 / 0.24	0.6793 / 0.51
有什么可以在家做的工作 What work can be done at home	0.7704 / 0.39	0.7848 / 0.46
有没有在家做兼职的工作? Are there any part-time jobs that can be done at home?	0.7321 / 0.46	0.7558 / 0.57
有什么工作在家就可以做 What work can be done at home	0.7736 / 0.25	0.7973 / 0.44
Ori. : 在家可以做的兼职有什么? 有什么 工 作是 必须 要 做 的? What are the part-time jobs that can be done at home? What work must be done? Adv.: 在家可以做的兼职有什么? 有什么 职 作是 固必 要 ? 的?		

Table 11: A generated adversarial example by TRAttack that successfully receives high $f(\cdot)$ / $s(\cdot)$ under 10 different queries meanwhil on the Baidu Cloud's commercial API.