TABS: Efficient Textual Adversarial Attack for Pre-trained NL Code Model Using Semantic Beam Search

YunSeok Choi, Hyojun Kim, Jee-Hyong Lee
College of Computing and Informatics
Sungkyunkwan University
Suwon, South Korea
{ys.choi, rlagywns0213, john}@skku.edu

Abstract

As pre-trained models have shown successful performance in program language processing as well as natural language processing, adversarial attacks on these models also attract attention. However, previous works on black-box adversarial attacks generated adversarial examples in a very inefficient way with simple greedy search. They also failed to find out better adversarial examples because it was hard to reduce the search space without performance loss. In this paper, we propose TABS, an efficient beam search black-box adversarial attack method. We adopt beam search to find out better adversarial examples, and contextual semantic filtering to effectively reduce the search space. Contextual semantic filtering reduces the number of candidate adversarial words considering the surrounding context and the semantic similarity. Our proposed method shows good performance in terms of attack success rate, the number of queries, and semantic similarity in attacking models for two tasks: NL code search classification and retrieval tasks.

1 Introduction

Many pre-trained models have been proposed and achieved success not only in natural language processing (Devlin et al., 2019; Liu et al., 2019; Sanh et al., 2019; Clark et al., 2020), but also in program language processing (Feng et al., 2020; Guo et al., 2021; Gotmare et al., 2021; Ahmad et al., 2021). As pre-trained models attract attention, so do the adversarial attacks on those models (Hsieh et al., 2019; Jin et al., 2020; Li et al., 2020; Garg and Ramakrishnan, 2020; Li et al., 2021). If pre-trained code models are attacked, they may produce incorrect results for slightly perturbed inputs in the sentence of natural language.

However, most of these black-box adversarial attack methods generate adversarial examples with inefficient greedy search. The method simply tries every adversarial word without considering the surrounding context and the contextual semantic word similarity. Moreover, it simply replaces vulnerable words in a sentence one by one without considering the global context. Their attack success rates are low, and a number of queries are required for attack success. It is not guaranteed that the adversarial examples are generated semantically similar to the original examples even if the attack was successful.

In this paper, we propose the semantic beam search method that can efficiently generate semantically similar adversarial examples with fewer queries while having higher performance. Our proposed method selects adversarial words considering the contextual semantic similarity with the original word as well as the word probability predicted by the BERT masked language model (MLM). With this contextual semantic filtering, the number of queries is significantly reduced compared to the existing methods without performance reduction. To increase the success rate, we adopt beam search (Wiseman and Rush, 2016) to generate better adversarial examples. Our approach keeps top-k candidate adversarial sentences with beam search while attacking. It helps choose a better adversarial sentence considering the global context. Beam search may expand the search space, but we can effectively reduce the search space with semantic filtering. Finally, we can achieve the highest success rate with a minimum number of queries.

We apply our approach to the NL-code search task that searches for codes with the same meaning for a given natural language sentence for the first time. The NL code search task provided by CodeXGLUE consists of two sub-tasks: classification task and retrieval task (Lu et al., 2021). The classification task is a task to classify whether or not the text-code pair represents the same meaning, and the retrieval task is a task that finds the most relevant code when a text is given. We show successful attack results for two tasks: classification and retrieval task of NL code search task.

2 Methodology

In this section, we describe TABS: Efficient Textual Adversarial Attack using Semantic Beam Search, that requires a very small number of queries to generate adversarial text with very high quality for the NL Code search tasks.

The proposed approach, TABS, is based on BERT-Attack. We determine the order of words to be replaced by output differences as the previous method did. Suppose that there is a pair of text $T = [w_1, w_2, ..., w_n]$ and code C, the corresponding label Y, and the output logit of the classification victim model or the output rank of the retrieval victim model, $f(T, C)$. The word importance by the output difference is defined as follows:

$$I_{w_i} = f(T, C) - f(T_{\backslash w_i}, C) \qquad (1)$$

where w_i is the i-th word in the sentence T, and $T \backslash w_i$ is the modified sentence that the word at the i-th position in T is replaced with <unk> token.

BERT-Attack obtains N synonym words for a word in T from the mask language model for a vulnerable word in the sentence. BERT-Attack replaces the word with each synonym word and checks whether the modified sentence is misclassified or not. If the attack is not successful even after it tried all the N synonyms, it chooses the synonym word with the lowest output logit values for the current word and continues to attack the next vulnerable word, which is a way of greedy search. When a word is being attacked, the previously attacked words are fixed in this greedy search, and the adversarial examples to be generated may be limited. Better synonym words for the previously attacked words can be found conditionally to the synonym of the current word, but the greedy search cannot change previously fixed synonyms.

To address this issue, we adopt beam search which keeps k candidate adversarial sentences while attacking. However, if beam search is simply applied, the number of queries may increase about k times. In order to solve this effectively, we do not try all the synonym words chosen by probability of the masked language model for the current position word, but we sample M important words among the N synonyms by considering the contextual semantic similarity to the original word.

Contextual semantic filtering is based on the similarity between word embeddings from BERT-MLM. First, we replace the current word with each synonym word obtained by probability of BERT-MLM in the sentence. We input the modified sentence into BERT-MLM once more, and obtain the contextual word embedding from the last hidden states of BERT-MLM. Since the embeddings are generated considering surrounding words, we can evaluate contextual semantic similarities between words. We consider not only the probability predicted by the masked language model but also the contextual semantic similarity to the original word for choosing synonym words, so we can effectively reduce the search space. We tokenize each word into sub-words using the Bytes-Pair-Encoding (BPE) which is the tokenizer of BERT-MLM. We use the average pooling to obtain the whole contextual word embedding of the original word.

Algorithm 1 TABS

```
Input: Text $T$; Code $C$; Victim model $f(\cdot)$
Output: Adversarial example $T^{adv}$
 1: for $w_i$ in $T$ do
 2:     Compute the importance score $I_{w_i}$ using Eq. 1
 3: end for
 4: Initialization $T^{adv} \leftarrow T$; $B \leftarrow \phi$; $B = B \cup T$; $B^{'} \leftarrow \phi$
 5: Create a word list $V$ sorted by descending order of importance score $I_{w_i}$
 6: for $v_i$ in $V$ do
 7:     for $b$ in $B$ do
 8:         Predict $N$ synonyms for $b \backslash v_i$ based on BERT-MLM
 9:         Calculate each context embedding for $N$ synonyms
10:         Sort by similarity scores based on context embedd-
11:         ings between the original word and $N$ synonyms
12:         Select top-$M$ words
13:         for $m$ in $M$ do
14:             $b[v_i] = m$
15:             $B^{'} \leftarrow B^{'} \cup b$
16:         end for
17:     end for
18:     Sort $B^{'}$ by the logit values of the Victim model $f(\cdot)$
19:     if Satisfy Eq. 2 or Eq. 3 in $B^{'}$ then
20:         $T^{adv} \leftarrow$ adversarial example in $B^{'}$ with the maximum sentence similarity to the original sentence $T$
21:         return $T^{adv}$
22:     else
23:         $B \leftarrow$ top-$K(B^{'})$, $B^{'} \leftarrow \phi$
24:     end if
25: end for
```

During attacking with beam search, we sort k candidate adversarial sentences by the logit values in ascending order, and check if there are adversarial sentences that satisfy the attack success condition. If there are, the sentence most similar to the original sentence is selected as the final adversarial example.

The attack success conditions are defined as fol-

Model	$\epsilon = 0.2$				$\epsilon = 0.3$			
	Success	**Perturb**	**Query**	**USE**	**Success**	**Perturb**	**Query**	**USE**
Textfooler (Jin et al., 2020)	699	15.9	67.5	0.466	578	15.1	67.8	0.511
BERT-Attack (Li et al., 2020)	792	13.9	84.2	0.474	667	13.1	83.7	0.514
CodeBERT-Attack	792	14.8	90.5	0.467	663	13.7	90.2	0.509
CLARE (Li et al., 2021)	814	**13.4**	923.1	0.484	689	**12.8**	969.7	0.525
Proposed-B	**882**	14.6	162.7	**0.507**	**787**	13.9	163.2	**0.537**
Proposed-S	780	15.2	**29.2**	0.461	634	14.3	**29.4**	0.509
Proposed-BS (TABS)	874	15.9	50.4	0.498	751	14.9	49.9	**0.537**

Table 1: Adversarial example generation performance in attack success (Success), change ratio (Perturb), number of queries (Query), and text similarity (USE) on the classification task of NL code search task. The original accuracy of the victim model on test dataset is 97.96%. The best result is in **boldface**, and the next best is underlined. Refer to Appendix B for the results of other ϵ.

Model	$\epsilon = 0.3$, $original\_rank \leq 100$, $target\_rank = 100$						
	Success	**Perturb**	**Rank**	**nDCG@10**	**nDCG@20**	**Query**	**USE**
Textfooler (Jin et al., 2020)	328	14.7	8.8/642.8	0.2033	0.2421	61.3	0.508
BERT-Attack (Li et al., 2020)	407	14.2	7.7/737.6	0.2007	0.2407	80.2	0.531
CLARE (Li et al., 2021)	395	**13.3**	8.0/725.9	0.2065	0.2496	907.9	0.537
Proposed-B	**461**	13.5	7.0/794.0	0.2097	0.2576	124.6	**0.557**
Proposed-S	405	15.0	8.0/730.7	0.1721	0.2129	**27.6**	0.516
Proposed-BS (TABS)	451	15.6	**7.2/800.3**	**0.1688**	**0.2130**	42.9	0.535

Table 2: Adversarial example generation performance in attack success (Success), change ratio (Perturb), original rank/changed rank (Rank), top-*10* nDCG (nDCG@10), top-*20* nDCG (nDCG@20), number of queries (Query), and text similarity (USE) on the retrieval task of NL code search task. The best result is in **boldface**, and the next best is underlined. The results of other $target\_rank$ value can be found in Appendix C.

lows:

$$\begin{aligned} &\operatorname{argmax}\left(f(T^{adv}, C)\right) \neq Y, \\ &\text{and } \operatorname{Sim}(T^{adv}, T) \geq \epsilon \end{aligned} \tag{2}$$

$$\begin{aligned} &f(T, C) < target\_rank < f(T^{adv}, C), \\ &\text{and } \operatorname{Sim}(T^{adv}, T) \geq \epsilon \end{aligned} \tag{3}$$

Eq. (2) is for classification task, and Eq. (3) for retrieval task. In the equations, T is an original sentence, T^{adv} is an adversarial sentence, C is a code snippet, $\operatorname{Sim}(\cdot, \cdot)$ is the cosine similarity between embedding vectors of the original and adversarial sentences, ϵ is the minimum similarity threshold between the original and adversarial sentences, and $target\_rank$ is the minimum rank for attack success.

We summarize the proposed approach, TABS, in Algorithm 1.

3 Experiments

3.1 Experiment Setting

We apply our approach on two NL code search tasks: classification and retrieval tasks (Lu et al., 2021). We fine-tune CodeBERT (Feng et al., 2020) for each of the two tasks. We evaluate our method on 1k test samples randomly selected from the test dataset.

As evaluation metrics, we use attack success, change ratio, number of queries, and text similarity in classification task, and attack success, change ratio, original rank/changed rank, nDCG (Wang et al., 2013), number of queries, and text similarity in retrieval task. Attack success is the number of successful attacks. Change ratio is on how many words are replaced for a successful attack, original rank is the rank of the correct code in the recommended code list by the code retrieval model, and changed rank is the rank by the attacked model. In the retrieval task, we need to consider not only the rank change of the correct code but also the change of recommended code list. We use Universal Sentence Encoder (USE) (Cer et al., 2018) to measure the text similarity between the original and adversarial sentences. We also observe nDCG to measure how much the top-k retrieved code list is changed by the adversarial text.

We use RoBERTa model (Liu et al., 2019) as BERT masked language model for our approach. We compare our approach with Textfooler, BERT-Attack, CodeBERT-Attack, and CLARE.

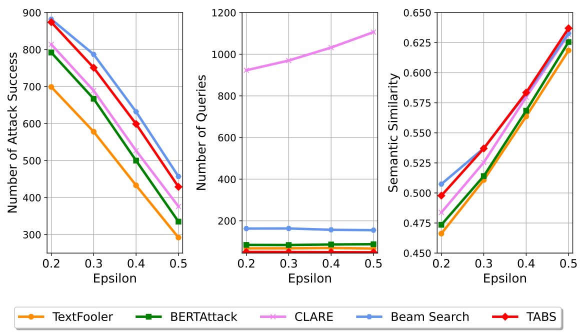

Figure 1: Comparison results of attack success, queries, and similarity on attacking CodeBERT fine-tuned for NL code search classification task with respect to ϵ. Our proposed method performs better than other baselines in all ϵ values.

Model	$\epsilon = 0.2$			
	Success	**Perturb**	**Query**	**USE**
BERT-Attack	792	**13.9**	84.2	0.474
TABS (5, 10)	874	15.9	50.4	0.498
TABS (5, 20)	878	15.1	80.5	0.508
TABS (10, 5)	857	16.8	**39.8**	0.491
TABS (10, 10)	**890**	16.0	73.5	**0.510**

Table 3: Performance on the combination of beam search candidate k and important synonym words m, (k, m).

CodeBERT-Attack is another type of BERT-Attack that uses the CodeBERT-MLM model for predicting synonym words. For all approaches, we set the hyper-parameter N to 50. Refer to the Appendix A for the NL code search dataset.

3.2 Experimental Result

Attack Result Table 1 shows the performance in the NL code search classification task. We compare the results at $\epsilon = 0.2$ and 0.3. Among the baselines, CLARE using various attack strategies, such as replace, insert, and merge, shows the highest attack success and USE. However, since CLARE calculates the importance score for 3 strategies to determine the order of attack words, it requires a large number of queries. BERT-Attack using BERT-MLM model and CodeBERT-Attack using CodeBERT-MLM show almost the same performance in the attack success rate, query, and USE.

Proposed-B is the proposed method with only beam search. It shows the performance of the highest semantic similarity compared to other baselines because it selects the sentence most similar to the original text among the candidates that satisfy the attack success condition. However, compared to the baselines such as BERT-Attack, Proposed-B has about twice as many queries. Proposed-S is the proposed method with only semantic filtering of synonyms by word embedding similarity. The attack success and USE are similar to BERT-Attack and higher than Textfooler, but the number of queries is significantly smaller than other baselines.

TABS (Proposed-BS), using both beam search and semantic filtering, maintains semantic similarity (USE) and attack success (Success) as high as Proposed-B, and significantly reduces the number of queries compared to other baselines. The result shows that our proposed method efficiently finds adversarial texts with semantic filtering and increases the quality of adversarial texts with beam search.

Table 2 shows the performance of the NL code search retrieval task. We filter out the test data with $original\_rank \leq 100$, which means the ground truth rank is less than 100 among 1k test samples, and we compare the results at $target\_rank = 100$ and $\epsilon = 0.3$ of the attack success condition. Similar to the results of the classification task, our approach shows significantly better performance in terms of attack success, the number of queries, USE, and rank change than the baseline models. The nDCG values are also the lowest compared to other baselines. It means that there are very few codes overlapped in the lists by the original text and the adversarial text.

Ablation Study We analyze the performance by changing the value of ϵ in the attack success condition. Figure 1 shows the results of attack success, the number of queries, and the semantic similarity with respect to ϵ in the classification task. Our approach shows superior performance to other baselines with any ϵ.

In addition, we analyze the performance with respect to k, the beam size of beam search, and m, the number of adversarial words chosen by the semantic filtering in Table 3. When m is doubled, the number of queries increases less than two times, and the success rate and USE increase slightly. The combination of $k = 10$ and $m = 10$ has better performance than $k = 5$ and $m = 20$. In comparison to BERT-Attack, all cases show better performance in success, query, and USE, and we select $k = 5$ and $m = 10$ that have good performance in three metrics.

Overall Efficiency The time complexity of BERT-Attack is $O(LN)$, and that of TABS is $O(LBM + LB \times mlm \times N)$ where L is sentence length, B is beam size, N is the number of synonyms, M is the number of important words, and mlm is BERT-MLM inference time complexity. We use $N = 50$, $B = 5$ and $M = 10$ so that $N = BM$. The time for contextual semantic filtering by BERT-MLM, $O(LB \times mlm \times N)$, is additionally required. However, it is possible to evaluate semantic similarity for N words at once if we use GPUs. If we build a batch with N modified sentences, we can evaluate their contextual semantic similarities with one inference of BERT-MLM. The actual time to calculate the similarity is very small compared to the overall time in TABS. It takes only about 3.9% of the overall average time in our experiment setting.

4 Conclusion

In this paper, we proposed an efficient beam search black-box adversarial attack method, TABS, that not only replaces words by considering the context of surrounding words but also reduces queries by considering the similarity of generated sentences. To the best of our knowledge, we are the first to apply an adversarial attack to the NL code search task. Our approach showed good performance in terms of attack success rate, number of queries, and semantic similarity in the classification task and attack success rate, number of queries, semantic similarity, rank change, and nDCG in the retrieval task.

Limitations

Although we are the first attempt at adversarial attack on the NL code search, our approach is needed to prove its applicability to other NL tasks, such as text classification and natural language inference, in future work.

Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00421, Artificial Intelligence Graduate School Program (Sungkyunkwan University), and No. 2022-0-01045, Self-directed Multi-modal Intelligence for solving unknown, open domain problems).

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668, Online. Association for Computational Linguistics.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, Brussels, Belgium. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Siddhant Garg and Goutham Ramakrishnan. 2020. BAE: BERT-based adversarial examples for text classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6174–6181, Online. Association for Computational Linguistics.

Akhilesh Deepak Gotmare, Junnan Li, Shafiq Joty, and Steven CH Hoi. 2021. Cascaded fast and slow models for efficient semantic code search. *ArXiv preprint*, abs/2110.07811.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcodebert: Pre-training code representations with data flow. In *9th International Conference on*

Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net.

Yu-Lun Hsieh, Minhao Cheng, Da-Cheng Juan, Wei Wei, Wen-Lian Hsu, and Cho-Jui Hsieh. 2019. On the robustness of self-attentive models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1520–1529, Florence, Italy. Association for Computational Linguistics.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *ArXiv preprint*, abs/1909.09436.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8018–8025.

Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. 2021. Contextualized perturbation for textual adversarial attack. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5053–5069, Online. Association for Computational Linguistics.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. BERT-ATTACK: Adversarial attack against BERT using BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv preprint*, abs/1910.01108.

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A theoretical analysis of NDCG type ranking measures. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, volume 30 of *JMLR Workshop and Conference Proceedings*, pages 25–54. JMLR.org.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

A Implementation Detail

We use the CodeSearchNet (Husain et al., 2019) dataset which is specialized for NL code search task provided by CodeXGLUE (Lu et al., 2021). We fine-tune CodeBERT for each of the two tasks.

Dataset	CodeSearchNet
Train	251,820
Valid	9,604
Test	19,210
Avg. Text Length	11
Avg. Code Length	95
Classification Accuracy	97.96%
Retrieval MRR	0.6796

Table 4: Statistics of CodeSearchNet dataset. Classification Accuracy and Retrieval MRR score are the performance of test dataset.

B Case Study on NL Classification Task

Model	$\epsilon = 0.4$				$\epsilon = 0.5$			
	Success	**Perturb**	**Query**	**USE**	**Success**	**Perturb**	**Query**	**USE**
Textfooler (Jin et al., 2020)	433	14.2	69.6	0.564	292	13.0	66.7	0.619
BERT-Attack (Li et al., 2020)	500	<u>12.3</u>	86.2	0.568	335	<u>11.2</u>	87.6	0.626
CodeBERT-Attack	486	12.9	92.0	0.567	312	11.9	93.0	0.631
CLARE (Li et al., 2021)	527	**12.1**	1032.1	<u>0.580</u>	376	**11.1**	1106.2	0.632
Proposed-B	**632**	12.8	156.8	**0.583**	**457**	11.9	155.0	<u>0.633</u>
Proposed-S	471	13.3	**30.6**	0.563	317	11.9	**30.8**	0.617
Proposed-BS (TABS)	<u>599</u>	13.7	<u>49.2</u>	**0.583**	<u>429</u>	12.6	<u>48.7</u>	**0.637**

C Case Study on Retrieval Task

Model	$\epsilon = 0.2$, $original\_rank \leq 100, target\, rank = 100$						
	Success	**Perturb**	**Rank**	**nDCG@10**	**nDCG@20**	**Query**	**USE**
Textfooler (Jin et al., 2020)	404	15.6	7.8/697.8	0.1803	0.2206	62.1	0.459
BERT-Attack (Li et al., 2020)	478	15.0	6.9/766.9	0.1852	0.2243	82.6	0.489
CLARE (Li et al., 2021)	462	**13.8**	7.2/762.5	0.1895	0.2329	880.2	0.495
Proposed-B	516	15.1	6.6/816.0	0.1809	0.2240	144.7	**0.525**
Proposed-S	514	16.1	6.8/837.1	0.1552	**0.1938**	**27.5**	0.460
Proposed-BS (TABS)	**525**	16.6	**6.5/841.9**	**0.1530**	0.1974	45.7	0.496

Model	$\epsilon = 0.3$, $original\_rank \leq 10, target\, rank = 100$						
	Success	**Perturb**	**Rank**	**nDCG@10**	**nDCG@20**	**Query**	**USE**
Textfooler (Jin et al., 2020)	259	15.6	2.1/599.0	0.1594	0.1953	64.7	0.493
BERT-Attack (Li et al., 2020)	333	14.8	2.0/741.6	0.1540	0.1984	84.2	0.512
CLARE (Li et al., 2021)	321	**13.8**	2.0/727.6	0.1588	0.2067	911.4	0.518
Proposed-B	**386**	15.1	2.0/784.9	0.1530	0.1979	148.9	**0.538**
Proposed-S	329	15.8	2.0/700.6	0.1337	**0.1753**	**28.9**	0.500
Proposed-BS (TABS)	378	16.4	**2.0/793.3**	**0.1262**	0.1758	47.1	0.513

Model	$\epsilon = 0.3$, $original\_rank = 1, target\, rank = 100$						
	Success	**Perturb**	**Rank**	**nDCG@10**	**nDCG@20**	**Query**	**USE**
Textfooler (Jin et al., 2020)	155	16.4	1.0/587.7	0.1142	**0.1485**	70.6	0.485
BERT-Attack (Li et al., 2020)	213	15.8	1.0/658.4	0.1255	0.1689	91.1	0.506
CLARE (Li et al., 2021)	198	**14.2**	1.0/619.1	0.1244	0.1741	929.8	0.510
Proposed-B	240	15.9	1.0/663.6	0.1322	0.1738	178.1	**0.537**
Proposed-S	209	16.6	1.0/733.1	0.1138	0.1524	**30.8**	0.498
Proposed-BS (TABS)	**241**	17.1	**1.0/762.2**	**0.1094**	0.1599	54.0	0.512

D Qualitative Examples

Code	def node_radius(self, node):
	    return self.get_idx(node) * self.scale + self.internal_radius

Ground Truth	Computes the radial position of the node .
Textfooler	Computes the radial position of the rope .
BERT-Attack	Computes the radial position of the compass .
CLARE	Computes the radial position of Z .
TABS	Computes the radial position of the vectors.

(a) Example 1

Code	def _build_state_value(request_handler, user):
	    uri = request_handler.request.url
	    token = xsrfutil.generate_token(xsrf_secret_key(), user.user_id(), action_id = str(uri))
	    return uri + ’:’ + token

Ground Truth	Composes the value for the state parameter .
Textfooler	Composes the value for the nations parameter .
BERT-Attack	Composes the value for the n parameter .
CLARE	Composes the value for the n parameter .
TABS	Composes the value for the computed parameter .

(b) Example 2

Code	def serialize_class (Cls):
	    metaclass = xtuml.get_metaclass(Cls)
	    attributes = [”%s %s’ % (name, ty.upper()) for name, ty in metaclass.attributes]
	    s = ’CREATE TABLE %s (\n’ % metaclass.kind
	    s += ’,\n’.join(attributes)
	    s += ’\n);\n’
	    return s

Ground Truth	Serialize an xtUML metamodel class .
Textfooler	Serialize an xtuml metamodel degrees .
BERT-Attack	Serialize an xtuml metamodel amide .
CLARE	Serialize an xtuml metamodel amide .
TABS	Serialize an xtuml metamodel id .

(c) Example 3