RobustQA: A Framework for Adversarial Text Generation Analysis on Question Answering Systems

Yasaman Boreshban[1*], **Seyed Morteza Mirbostani**[2*],
Seyedeh Fatemeh Ahmadi[2], **Gita Shojaee**[2], **Fatemeh Kamani**[2],
Gholamreza Ghassem-Sani[1], and **Seyed Abolghasem Mirroshandel**[2]
[1]Computer Engineering Department, Sharif University of Technology, Tehran, Iran
[2]Computer Engineering Department, University of Guilan, Rasht, Iran
{yasaman.boreshban, sani}@sharif.edu
m.mirbostani@msc.guilan.ac.ir mirroshandel@guilan.ac.ir

Abstract

Question answering (QA) systems have reached human-level accuracy; however, these systems are not robust enough and are vulnerable to adversarial examples. Recently, adversarial attacks have been widely investigated in text classification. However, there have been few research efforts on this topic in QA. In this article, we have modified the attack algorithms widely used in text classification to fit those algorithms for QA systems. We have evaluated the impact of various attack methods on QA systems at character, word, and sentence levels. Furthermore, we have developed a new framework, named RobustQA, as the first open-source toolkit for investigating textual adversarial attacks in QA systems. RobustQA consists of seven modules: Tokenizer, Victim Model, Goals, Metrics, Attacker, Attack Selector, and Evaluator. It currently supports six different attack algorithms. Furthermore, the framework simplifies the development of new attack algorithms in QA.

1 Introduction

With the release of large and high-quality datasets in the field of question answering (QA) (Rajpurkar et al., 2016; Joshi et al., 2017; Trischler et al., 2017; Kočiský et al., 2018), we witness significant progress in this area of research. With the aid of deep neural networks (DNNs), the newly presented models have even reached human-level accuracy. However, it has been shown that these models are not yet robust enough and are vulnerable to adversarial examples (Gil et al., 2019; Ren et al., 2019).

In the context of QA systems, the model's accuracy drops drastically when some adversarial sentences are added to the input paragraphs (Jia and Liang, 2017). Accordingly, extensive research efforts have been conducted addressing various techniques to increase the robustness of DNN models in different fields. One of the most popular techniques to overcome this issue is the so-called adversarial training. In adversarial training, some adversarial examples are used during the training phase of the model to increase its robustness against textual adversarial attacks (Jia and Liang, 2017; Gan and Ng, 2019). In another research, the impact of the knowledge distillation technique on the robustness of QA models has been analyzed (Boreshban et al., 2023).

Adversarial attacks have been widely investigated in the field of text classification (Li et al., 2020; Jin et al., 2020). Furthermore, OpenAttack (Zeng et al., 2021) and TextAttack (Morris et al., 2020) frameworks have been presented to simplify the implementation and analysis of different attack methods in text classification. However, there has been only a limited number of efforts in this regard for the QA systems.

The contributions of the paper can be summarized as follows: 1) We modify the attack algorithms that have been widely used in the field of text classification for QA systems. 2) We show that these modified attack algorithms can easily be evaluated on QA systems in three different characters, words, and sentence levels. 3) We build a new open-source framework named RobustQA, aiming at simplifying the research on textual adversarial attacks in QA systems. 4) We have incorporated both adversarial text generation and data augmentation in RobustQA for being used in adversarial training methods to improve the robustness and generalization of QA models.

In this paper, we introduce the related works in Section 2. We compare the QA task against text classification and describe a sample textual adversarial attack algorithm implemented for the QA task in Section 3. Next, we introduce the RobustQA framework modules in detail in Section 4 and demonstrate the framework's usage in Section 5. We present our setup and experimental

[*]These authors contributed equally to this work.

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 274–285
December 6-10, 2023

results in Section 6. Finally, our conclusions and future works are presented in Section 7.

2 Related works

2.1 Adversarial Sentences in Text Classification

Adversarial attacks have been extensively studied on continuous data (Goodfellow et al., 2014; Moosavi-Dezfooli et al., 2017); however, addressing these attacks on discrete data such as text (Xu et al., 2020; Zhang et al., 2020) poses significant challenges.

Adversarial attacks can be categorized based on different aspects. Attacks are primarily divided into two types of white and black boxes. In white box attacks, the attacker has full access to the model and its parameters. In this type of attack, the gradient of the cost function relative to the input is used to generate an adversarial example (Papernot et al., 2016; Ebrahimi et al., 2018; Li et al., 2018; Wallace et al., 2019). In the black box attack, however, there is limited knowledge regarding the model, and thus one can only use the output of the model to generate an adversarial example (Jin et al., 2020; Garg and Ramakrishnan, 2020; Li et al., 2020).

Adversarial attacks are also divided into untargeted and targeted categories. In untargeted attacks, the goal is merely to cause the model to produce an incorrect output label (Pruthi et al., 2019; Garg and Ramakrishnan, 2020); whereas in targeted ones, more restrictions are applied to impose a specific wrong prediction (Gao et al., 2018; Wang et al., 2020).

Textual adversarial attacks are divided into three categories in terms of perturbation levels, i.e., character, word, and sentence. Character-level attacks usually manipulate characters based on insertion, deletion, swap, substitution, and repetition operations (Gao et al., 2018; Eger et al., 2019; Gil et al., 2019; He et al., 2021). In word-level attacks, words are replaced with their synonyms. In this case, the algorithms consist of two stages. At first, resources such as the word embedding (Jin et al., 2020), language models (Li et al., 2020; Garg and Ramakrishnan, 2020), and semantic networks (Ren et al., 2019) are used to produce a set of perturbations. In the second stage, using different algorithms such as greedy search (Li et al., 2018; Ren et al., 2019), beam search (Ebrahimi et al., 2018), genetic algorithm (Alzantot et al., 2018), and particle swarm optimization (Zang et al., 2020), successful queries are selected. Finally, in sentence-level attacks, special techniques such as adding misleading sentences to the text (Jia and Liang, 2017), paraphrasing (Iyyer et al., 2018; Ribeiro et al., 2018; Gan and Ng, 2019; Huang and Chang, 2021), and using the autoencoder structure (Zhao et al., 2017; Wang et al., 2020) are employed to produce adversarial sentences.

2.2 Adversarial Sentences in QA Systems

There have been only a few research initiatives focused on textual adversarial attacks in the field of QA.

Jia and Liang (2017) showed that QA systems get confused by appending misleading sentences to the input paragraph. They introduced two algorithms called *AddSent* and *AddAny*. Later, Yang et al. (2021) improved these algorithms by introducing *AddSentDivers* to increase the diversity of the generated adversarial sentences.

It has been demonstrated that paraphrasing the questions is an alternative method for generating adversarial sentences. In this regard, Ribeiro et al. (2018) used the back translation technique to obtain question paraphrase rules. Also, Gan and Ng (2019) used a transformer model to produce paraphrased questions and introduced two types of adversarial questions. The autoencoder structure was utilized in another recent research to generate adversarial sentences (Wang et al., 2020).

2.3 Available Frameworks

There are several open-source libraries for building adversarial examples on continuous data. The most notable ones are CleverHans (Papernot et al., 2016), Foolbox (Rauber et al., 2017), Adversarial Robustness Toolbox (Nicolae et al., 2018), and AsvBox (Goodman et al., 2020). On the contrary, a limited number of open-source libraries operable on the discrete data are available. SeqAttack (Simoncini and Spanakis, 2021) is a framework for conducting adversarial attacks on the named entity recognition problems. The most famous frameworks for creating adversarial sentences in text classification are OpenAttack (Zeng et al., 2021) and TextAttack (Morris et al., 2020). Although these frameworks are suitable for text classification, these algorithms do not currently support QA systems.

3 Question Answering vs. Text Classification

3.1 Task Structure

In QA systems, the question and context are represented as a sequence of tokens, $\mathbf{Q} = \{q_1, q_2, q_3, ..., q_n\}$ and $\mathbf{C} = \{c_1, c_2, c_3, ..., c_n\}$, respectively. In these systems, the main goal is to predict the answer, $\mathbf{A}$, in the form of a span within the context, $\mathbf{A} = \{c_j, ..., c_{j+k}\}$. The returned span includes a specific start and an end token indices of the context paragraphs. F1 score and exact match (EM) criteria are the two common metrics for evaluating QA systems.

On the other hand, in the text classification task, the main goal is to recognize the correct class of an input text. Due to the substantial differences between QA and text classification tasks, the algorithms designed for dealing with the attacks in the text classification are not directly applicable to the QA problems. The main distinctions are related to their differences in the structure of the input data and the goal function of attack scenarios.

3.2 Input Data Structure

In text classification of an input text $\mathbf{X}$ with the corresponding ground truth label Y and the victim model F, the goal of an attack scenario is to have an attack set up that transforms $\mathbf{X}$ to $\tilde{\mathbf{X}}$ with the minimum perturbation in such a way that the victim model predicts an incorrect label $\tilde{Y}$, where $Y \neq \tilde{Y}$.

In QA tasks, every input $\mathbf{X}$ is composed of a question $\mathbf{Q}$ and a context $\mathbf{C}$.

$$\mathbf{X} = x_1 x_2 \cdots x_i \cdots x_n, \quad x_i \in \{\mathbf{Q}, \mathbf{C}\} \tag{1}$$

The ground truth label $\mathbf{Y}$, which is a part of the given context with specific start and end tokens, represents the correct answer to the given question.

$$\mathbf{Y} = c_j \cdots c_{j+k} \quad c_j \in \{\mathbf{C}\} \tag{2}$$

The predicted answer $\tilde{\mathbf{Y}}$ is computed by considering the maximum probability for the start and end tokens.

$$\tilde{\mathbf{Y}} = \mathrm{F}(\arg\max_{x \in \mathbf{C}} \mathrm{P}(x)) \tag{3}$$

Akin to the text classification, the goal of an attack scenario here is to have an attack set up that transforms $\mathbf{X}$ to $\tilde{\mathbf{X}}$ with the minimum perturbation Δx in a way that the victim model predicts an incorrect answer span $\tilde{\mathbf{Y}}$, where

$$\tilde{x} = x + \Delta x, \quad ||\Delta x||_p < \epsilon \tag{4}$$

$$\tilde{\mathbf{Y}} \neq \mathbf{Y} \tag{5}$$

3.3 Goal Function Criteria

In both text classification and QA tasks, the goal function of an attack scenario determines the success of the attack on a given victim model. In text classification, an attack scenario for a given input example is regarded as successful if the model prediction for the example is not equal to its corresponding ground truth label. In this task, the goal function can be simply evaluated by a single criterion.

However, in QA tasks, since a prediction label includes two items (i.e., the start and the end tokens of the predicted answer span), the goal function is usually evaluated by both F1 and EM criteria.

3.4 Attack Methods

To demonstrate the required modifications of an attack method to cope with the mentioned differences, we discuss the details of the changes applied to the TextFooler algorithm (Jin et al., 2020). We have applied similar changes to few other attack algorithms in the new framework to make those algorithms fit for QA tasks.

The TextFooler algorithm is a score-based textual adversarial attack that consists of two primary steps. The first step is the *word importance ranking*, in which words are sorted according to their importance. The second step is the *word transformation*, which produces suitable substitutes for the words with the highest importance level obtained from the first step to generate an adversary example.

Algorithm 1 shows the pseudo-code of a revised version of the TextFooler algorithm, which is compatible with QA tasks.

Word Importance Ranking (line 1-11) The input example $\mathbf{X}$, which includes context $\mathbf{C}$ and question $\mathbf{Q}$, accompanied by its corresponding ground truth label $\mathbf{Y}$, is passed to the algorithm. The goal is to confuse the victim model by generating a new question $\tilde{\mathbf{Q}}$, with the minimum perturbation to $\mathbf{Q}$. One metric among the F1 score and EM measure is used for marking an adversarial example. In Algorithm 1, we use the F1 score and δ, a threshold value empirically set to 0.9, as the goal function criterion.

Algorithm 1: QA Adversarial Attack by TextFooler

Input: Input example $\mathbf{X} = \{\mathbf{Q}, \mathbf{C}\} = \{w_1, w_2, \cdots, w_n\}$, the corresponding ground truth label $\mathbf{Y}$, victim model F, victim model's prediction P, sentence similarity function $\mathrm{Sim}(\cdot)$, sentence similarity threshold ϵ, word embeddings Emb over the vocabulary Vocab, F1 score function $\mathrm{F1}(\cdot)$, goal's F1 score threshold δ, adversarial question $\tilde{\mathbf{Q}}$

Output: Adversarial example $\tilde{\mathbf{X}} = \{\tilde{\mathbf{Q}}, \mathbf{C}\}$

Initialization: $\tilde{\mathbf{X}} \leftarrow \mathbf{X}$
for each word w_i in $\mathbf{Q}$ **do**
  Compute the importance score of the start and end answer span, $I_{w_i} = (I^s_{w_i}, I^e_{w_i})$
  if $\mathrm{F}(\mathbf{X}) = \mathrm{F}(\mathbf{X}_{\setminus w_i}) = \mathbf{Y}$ **then**
    $(I^s_{w_i}, I^e_{w_i}) \leftarrow \mathrm{P}_{\mathbf{Y}}(\mathbf{X}) - \mathrm{P}_{\mathbf{Y}}(\mathbf{X}_{\setminus w_i})$
  else if $\mathrm{F}(\mathbf{X}) = \mathbf{Y}$, $\mathrm{F}(\mathbf{X}_{\setminus w_i}) = \bar{\mathbf{Y}}$, and $\mathbf{Y} \neq \bar{\mathbf{Y}}$ **then**
    $(I^s_{w_i}, I^e_{w_i}) \leftarrow (\mathrm{P}_{\mathbf{Y}}(\mathbf{X}) - \mathrm{P}_{\mathbf{Y}}(\mathbf{X}_{\setminus w_i})) + (\mathrm{P}_{\bar{\mathbf{Y}}}(\mathbf{X}_{\setminus w_i}) - \mathrm{P}_{\bar{\mathbf{Y}}}(\mathbf{X}))$
  end
end
Create a set $\mathbf{W}$ of all words $w_i \in \mathbf{Q}$ sorted by the descending order of their importance score, either using start $I^s_{w_i}$ or average $(I^s_{w_i} + I^e_{w_i})/2$ importance score.
Filter out the stop words in $\mathbf{W}$.
for each word w_j in $\mathbf{W}$ **do**
  Initiate the set of candidates CANDIDATES by extracting the top N synonyms using $\mathrm{CosSim}(\mathrm{Emb}_{w_j}, \mathrm{Emb}_{\text{word}})$ for each word in Vocab.
  CANDIDATES $\leftarrow \mathrm{POSFilter}(\text{CANDIDATES})$
  FINCANDIDATES $\leftarrow \{\}$
  for c_k *in* CANDIDATES **do**
    $\mathbf{X}' \leftarrow$ Replace w_j with c_k in $\tilde{\mathbf{X}}$
    if $\mathrm{Sim}(\mathbf{Q}', \tilde{\mathbf{Q}}) > \epsilon$ **then**
      FINCANDIDATES $\leftarrow$ FINCANDIDATES $\cup \{c_k\}$
      $\mathbf{Y}_k \leftarrow \mathrm{F}(\mathbf{X}')$
      $\mathbf{P}_k \leftarrow \mathrm{F}_{\mathbf{Y}_k}(\mathbf{X}')$
    end
  end
  $\alpha = (\mathrm{F1}(\mathbf{X}) - \mathrm{F1}(\mathbf{X}'))/\mathrm{F1}(\mathbf{X})$
  if there exists c_k where $\alpha > \delta$ **then**
    In FINCANDIDATES, only keep the candidates c_k where $\alpha > \delta$
    $c^* \leftarrow \underset{c \in \text{FINCANDIDATES}}{\mathrm{argmax}} \mathrm{Sim}(\mathbf{Q}, \mathbf{Q}'_{w_j \to c})$
    $\tilde{\mathbf{Q}} \leftarrow$ Replace w_j with c^* in $\tilde{\mathbf{Q}}$
    return $\{\tilde{\mathbf{Q}}, \mathbf{C}\}$
  else if $\underset{c_k \in \text{FINCANDIDATES}}{\min} \mathbf{P}_k < \mathrm{P}_{\mathbf{Y}_k}(\tilde{\mathbf{X}})$ **then**
    $c^* \leftarrow \underset{c_k \in \text{FINCANDIDATES}}{\mathrm{argmin}} \mathbf{P}_k$
    $\tilde{\mathbf{Q}} \leftarrow$ Replace w_j with c^* in $\tilde{\mathbf{Q}}$
  end
end
return None

First, a copy of $\mathbf{X}$ is taken as a potential adversarial example $\tilde{\mathbf{X}}$. Then, question $\mathbf{Q}$ is systematically altered for n number of times by in turn deleting the token w_i. Each altered question is then passed to the victim model to predict the answer span based on the highest probability values of the start and end tokens. Next, the model predictions for the initial question (i.e., $\mathbf{Y}$) and that of each altered question (i.e., $\bar{\mathbf{Y}}$) are compared. Accordingly, the importance score of the start and end tokens of the answer span $I_{w_i} = (I^s_{w_i}, I^e_{w_i})$ for each altered question is computed by either line 5 (i.e., in the case of equality) or line 7 (i.e., otherwise).

$\mathrm{P}_{\mathbf{Y}}(\mathbf{X})$ and $\mathrm{P}_{\bar{\mathbf{Y}}}(\mathbf{X})$ respectively represent the probability values of the start and end tokens of the answer span provided by the ground truth label $\mathbf{Y}$ and that of the label $\bar{\mathbf{Y}}$ predicted by the attacked model for the original question $\mathbf{X}$. Similarly, $\mathrm{P}_{\mathbf{Y}}(\mathbf{X}_{\setminus w_i})$ and $\mathrm{P}_{\bar{\mathbf{Y}}}(\mathbf{X}_{\setminus w_i})$ respectively represent the probability values of the start and end tokens of the answers predicted by the original and attacked model for the perturbed question, in which $\mathbf{w_i}$ has been omitted from the original question.

In line 10, a set of $\mathbf{W}$ of all words $w_i \in \mathbf{Q}$ is created and sorted by the descending order of their importance score (i.e., using $I^s_{w_i}$ or $(I^s_{w_i} + I^e_{w_i})/2$). In our experiments, we have chosen $I^s_{w_i}$ to compute the importance score.

Word Transformation (line 12-34) In lines 12-14, using the Cosine similarity metric, a set of candidates CANDIDATES is created by extracting the top N synonyms of word w_j with the same part of speech as that of w_j.

In lines 15 to 23, each word w_j is in turn substituted by a candidate (i.e., c_k) to create an altered example (i.e., $\mathbf{X}'$). Among all the candidates, those that cause the similarity between the potential adversarial question (i.e., $\tilde{\mathbf{Q}}$) and the altered question (i.e., $\mathbf{Q}'$) to exceed a predefined threshold (that we empirically set it to 0.7), are considered as final candidates. Each final candidate along with its predicted label (i.e., $\mathbf{Y}_k$) and the probability values of its start and end tokens (i.e., $\mathbf{P}_k$) are stored.

In lines 24-33, at first, the eligibility of each final candidate as an adversarial example is determined by computing an α value for the candidate and comparing the value against a predefined threshold δ. If a candidate modifies the initial question $\mathbf{Q}$ in a way that results in an altered question $\mathbf{Q}'$ having the maximum semantic similarity with $\mathbf{Q}$, then $\mathbf{Q}'$ will be chosen as an adversarial question. However, if a candidate does not satisfy this condition, one of the final candidates with the least confidence score is instead selected.

In RobustQA, we modified TextFooler (Jin et al., 2020), VIPER (Eger et al., 2019; He et al., 2021), Genetic (Alzantot et al., 2018), BERT Attack (Li et al., 2020), PWWS (Ren et al., 2019), SememePSO (Zang et al., 2020), TextBugger (Li et al., 2018), SCPN (Iyyer et al., 2018), and DeepWordBug (Gao et al., 2018) algorithms to be compatible with the QA systems. Note that all the mentioned modifications preserve the nature of these attack algorithms.

4 The RobustQA Framework

We have developed a new attack framework named RobustQA for applying text adversarial attack algorithms to QA systems. This framework is an extension of OpenAttack (Zeng et al., 2021), which has been designed for implementing text classification adversarial attacks. RobustQA consists of seven modules, depicted in Figure A.1:

Tokenizer. The tokenizer module of RobustQA supports multiple tokenization approaches, including word-, sub-word-, and character-level tokenization. It maintains the consistency between the tokenization of the original sample and that of the adversarial one, enabling the effective evaluation of the attack algorithms. Furthermore, it currently supports the Stanford question answering dataset (SQuAD) dataset (Rajpurkar et al., 2016). However, it can be extended to support any other QA datasets, such as TriviaQA (Joshi et al., 2017), NewsQA (Trischler et al., 2017), etc.

Victim Model. The victim model module supports the QA-based models. An extended version of this module is implemented to integrate HuggingFace Transformer-based models[*]. This module contains multiple methods required for executing different adversarial attack scenarios in RobustQA. These methods can be overridden or extended for any desired customized attack, as they have access to all the sub-layers of the model's output and perform their operation as middleware.

Goals. The primary target of the goal module is to determine if an input sample is eligible as an adversarial candidate. The candidate sample is regarded as an eligible one if it can confuse the victim model and diminish its performance in terms of EM or F1 score metrics. Defining a custom goal

[*]https://huggingface.co/

for new QA attacks is achievable by extending the goal module.

Metrics. The evaluation metrics of the attack scenarios can be selected or extended with this module. As discussed in Appendix B, the evaluation metrics specific to the QA task (i.e., EM and F1 score) are enabled by default. Other metrics such as edit distance, fluency, grammatical errors, modification rate, and semantic similarity are available for selection.

Attacker. The attacker is an abstract module with a default implementation of all the required tools and logic to define an attack algorithm on a given QA victim model. Based on the F1 score metric and a predefined threshold value, an attack goal specific to the given QA task is defined and used as a criterion to determine the adversary potential of different input examples. The primary method of creating a custom QA attack algorithm is to extend the QA attacker module. Various types of adversarial attack algorithms are derived from this module in RobustQA, ready for experimentation.

Attack Selector. The attack selector module facilitates the initiation of an attack scenario. This module enables effortless selection and instantiation of the victim model, tokenizer, dataset, attacker, and evaluation metrics. It also performs data sampling and preparation. An attack scenario is easily configured by passing the preferred settings to the attack selector module. Further comprehensive analysis of the attack algorithms is possible by providing additional customized metrics to this module.

Evaluator. The execution and evaluation of the QA attack algorithms take place in the evaluator module. Attacks performance is evaluated from different aspects: (1) the attack success rate indicates the percentage of the attacks that fool the victim model and produce false predictions; (2) the modification rate is the percentage of the modified tokens in an adversarial example compared to the input example; (3) the fluency of adversarial examples are computed by perplexity by GPT-2 (Radford et al., 2019); (4) the grammatical errors of each adversarial example is compared to that of the original example, using an available language tool; (5) the semantic similarity between an input example and an adversarial example is computed using a universal sentence encoder (Cer et al., 2018); and finally (6) the average time devoted to the query and attack execution is used to measure the efficacy of different attacks.

5 Toolkit Usage

The RobustQA interface empowers users to execute attack scenarios either programmatically, utilizing the Python programming language, or via a command-line prompt. Appendix D demonstrates an example of the toolkit usage through command-line interface and code. Moreover, some adversarial examples generated by different attack algorithms are depicted in Appendix E.

6 Experiments

Utilizing RobustQA, we have evaluated the performance of six different adversarial attack algorithms on the large uncased Bidirectional Encoder Representations from Transformer (BERT) model (Devlin et al., 2019) using the SQuAD dataset, explained in Appendix C. Moreover, we have augmented the training set of SQuAD with an additional 10% adversarial examples generated through the BertAttack algorithm to evaluate the robustness of a given victim model trained by the adversarial training technique. In these experiments, we have considered multiple metrics to evaluate the quality of generated adversarial examples. The results of our experiments with RobustQA and our system setup are presented in Appendices F and G, respectively.

7 Conclusion

In this article, we showed the effect of various textual adversarial attack algorithms in character, word, and sentence levels on QA systems. We also developed an open-source framework, named RobustQA, for the field of textual adversarial attack on QA systems, which consists of seven primary modules. This new framework offers different features that are easily customizable for applying existing or designing new algorithms, along with efficient analysis of attack scenarios. As our future work, this framework can be further extended to include other attack algorithms. We can also provide more functions and tools for further research in the context of attacks and defense within QA systems. The source code and documentation of RobustQA are available at `https://github.com/mirbostani/RobustQA`.

Limitations

Although RobustQA is reliable for implementing and evaluating textual adversarial attacks on QA models, a limitation may arise in certain attack algorithms due to their high resource requirements. Specifically, in some cases, the execution of the attack algorithms requires a high level of GPU resources and CPU iterations. Like many other deep learning algorithms, adversarial text generation and adversarial training heavily rely on GPU resources. As the augmented training set grows, the mentioned procedures demand a substantial share of GPU power. This requirement imposed some constraint on the extent of our experiments.

Due to the intricacies of the QA domain and the diverse nature of attacks in this domain, it was not feasible for us to seamlessly integrate all of them. Some algorithms could perfectly align with specific QA architectures, while others might require some customizations. Although the required tools for implementing any adversarial attack algorithm can be embedded within the RobustQA framework, the challenge of adapting all the attack algorithms hindered the variety of our experiments conducted in this study.

Ethics Statement

The primary focus of this study has been on enhancing the robustness of NLP models to make these models less vulnerable to potential misusage. We foresee no ethical issues arising from the algorithms and techniques introduced in this study. All the datasets, tools, and libraries employed in this study are open-source and publicly accessible.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium. Association for Computational Linguistics.

Yasaman Boreshban, Seyed Morteza Mirbostani, Gholamreza Ghassem-Sani, Seyed Abolghasem Mirroshandel, and Shahin Amiriparian. 2023. Improving question answering performance using knowledge distillation and active learning. *Engineering Applications of Artificial Intelligence*, 123:106137.

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.

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.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia. Association for Computational Linguistics.

Steffen Eger, Gözde Gül Şahin, Andreas Rücklé, Ji-Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, and Iryna Gurevych. 2019. Text processing like humans do: Visually attacking and shielding NLP systems. 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 1634–1647, Minneapolis, Minnesota. Association for Computational Linguistics.

Wee Chung Gan and Hwee Tou Ng. 2019. Improving the robustness of question answering systems to question paraphrasing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6065–6075, Florence, Italy. Association for Computational Linguistics.

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)*, pages 50–56. IEEE.

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.

Yotam Gil, Yoav Chai, Or Gorodissky, and Jonathan Berant. 2019. White-to-black: Efficient distillation of black-box adversarial attacks. 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 1373–1379, Minneapolis, Minnesota. Association for Computational Linguistics.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Dou Goodman, Hao Xin, Wang Yang, Wu Yuesheng, Xiong Junfeng, and Zhang Huan. 2020. Advbox: a toolbox to generate adversarial examples that fool neural networks. *arXiv preprint arXiv:2001.05574*.

Xuanli He, Lingjuan Lyu, Qiongkai Xu, and Lichao Sun. 2021. Model extraction and adversarial transferability, your bert is vulnerable! *arXiv preprint arXiv:2103.10013*.

Kuan-Hao Huang and Kai-Wei Chang. 2021. Generating syntactically controlled paraphrases without using annotated parallel pairs. *arXiv preprint arXiv:2101.10579*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark. Association for Computational Linguistics.

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, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

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. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202, Online. Association for Computational Linguistics.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773.

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, Online. Association for Computational Linguistics.

Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, et al. 2018. Adversarial robustness toolbox v1. 0.0. *arXiv preprint arXiv:1807.01069*.

Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, et al. 2016. Technical report on the cleverhans v2. 1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*.

Danish Pruthi, Bhuwan Dhingra, and Zachary C. Lipton. 2019. Combating adversarial misspellings with robust word recognition. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5582–5591, Florence, Italy. Association for Computational Linguistics.

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.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. 2017. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*.

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, Florence, Italy. Association for Computational Linguistics.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, Melbourne, Australia. Association for Computational Linguistics.

Walter Simoncini and Gerasimos Spanakis. 2021. SeqAttack: On adversarial attacks for named entity recognition. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 308–318, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada. Association for Computational Linguistics.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.

Boxin Wang, Hengzhi Pei, Boyuan Pan, Qian Chen, Shuohang Wang, and Bo Li. 2020. T3: Tree-autoencoder constrained adversarial text generation for targeted attack. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6134–6150, Online. Association for Computational Linguistics.

Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K Jain. 2020. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178.

Ziqing Yang, Yiming Cui, Chenglei Si, Wanxiang Che, Ting Liu, Shijin Wang, and Guoping Hu. 2021. Adversarial training for machine reading comprehension with virtual embeddings. In *Proceedings of *SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 308–313, Online. Association for Computational Linguistics.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online. Association for Computational Linguistics.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Zixian Ma, 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, Online. Association for Computational Linguistics.

Wei Emma Zhang, Quan Z Sheng, Ahoud Alhazmi, and Chenliang Li. 2020. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(3):1–41.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2017. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342*.

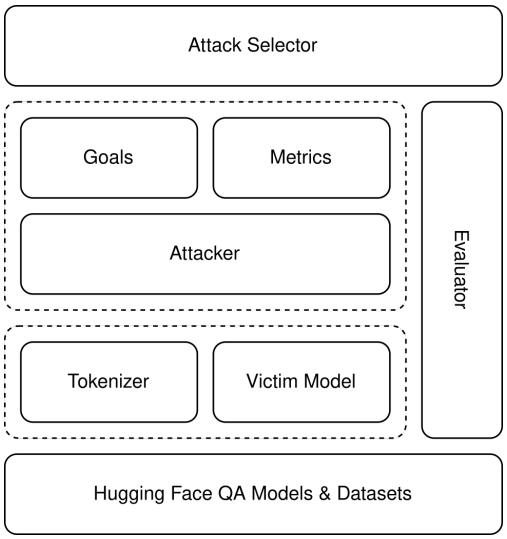

Figure A.1: The architecture of RobustQA.

A Architecture

RobustQA is the first open-source framework for textual adversarial attack analysis in QA systems. As shown in Figure A.1, it consists of seven modules: Tokenizer, Victim Model, Goals, Metrics, Attacker, Attack Selector, and Evaluator. Currently, six different adversarial attack algorithms have been implemented in this framework.

B Evaluation Metrics

For the evaluation purpose, we have employed EM and F1 score criteria, which are regarded as the standard metrics for evaluating QA systems (Rajpurkar et al., 2016). The F1 measure represents the average overlap between the ground truth and the predicted answers. On the other hand, the EM measure demonstrates the percentage of those responses that exactly match the ground truth answer.

```
MODEL="bert-large-uncased-whole-word-masking\
-finetuned-squad"
python qa.py \
    --use_cuda \
    --victim_model_or_path "$MODEL" \
    --victim_tokenizer_or_path "$MODEL" \
    --dataset "squad" \
    --dataset_split "validation[0:1000]" \
    --attack_recipe "textfooler" \
    --batch_size 8 \
    --language "english" \
    --use_metric_f1_score \
    --use_metric_exact_match \
    --use_metric_edit_distance \
    --use_metric_fluency \
    --use_metric_grammatical_errors \
    --use_metric_modification_rate \
    --use_metric_semantic_similarity \
    --use_metric_jaccard_char_similarity \
    --use_metric_jaccard_word_similarity
```

Figure D.1: Executing the TextFooler attack on $\text{BERT}_{\text{LARGE}}$ via command-line interface.

```
+=========================================+
|                Summary                  |
+=========================================+
| Total Attacked Instances:       | 1000  |
| Successful Instances:           | 230   |
| Attack Success Rate:            | 0.23  |
| Avg. Running Time:              | 0.30072 |
| Total Query Exceeded:           | 0     |
| Avg. Victim Model Queries:      | 85.819 |
| Avg. Exact Match (Orig):        | 57.1  |
| Avg. Exact Match (Adv):         | 46.2  |
| Avg. F1 Score (Orig):           | 72.305 |
| Avg. F1 Score (Adv):            | 59.183 |
| Avg. Levenshtein Edit Distance: | 1.6609 |
| Avg. Fluency (ppl):             | 1765.1 |
| Avg. Grammatical Errors:        | 0     |
| Avg. Word Modif. Rate:          | 0.15899 |
| Avg. Semantic Similarity:       | 0.84477 |
| Avg. Jaccard Char Similarity:   | 0.91577 |
| Avg. Jaccard Word Similarity:   | 0.46207 |
+=========================================+
```

Figure D.2: The results of the TextFooler attack on $\text{BERT}_{\text{LARGE}}$.

C Datasets

The SQuAD v1.1, introduced in 2016 by (Rajpurkar et al., 2016), is a reading comprehension dataset containing 107,785 question-answer pairs derived from 536 Wikipedia documents. In this version of SQuAD, the answer to each question is a span of the text extracted from the associated paragraph in the document. The training and validation datasets contain 87,599 and 10,570 question-answer pairs, respectively.

D Usage Examples

Figure D.1 is an example demonstrating the usage of the RobustQA framework through the command-line interface. The TextFooler attack algorithm is executed on the $\text{BERT}_{\text{LARGE}}$ model employing the first 1000 validation examples of the SQuAD dataset.

The results of the TextFooler attack on the $\text{BERT}_{\text{LARGE}}$ model along with all the computed metric values are summarized in Figure D.2.

The same attack scenario can be executed by code using the Python programming language demonstrated in Figure D.3.

E Adversarial Examples

In this section, the generated adversarial examples of three attack algorithms are presented. The original and adversary questions are depicted in Table E.1. Other fields of the generated adversarial examples, such as "context" and "answers", are the same as the original instance from the SQuAD dataset.

F Results

In this section, we present the evaluation results of six different adversarial attack algorithms implemented with the RobustQA framework. The experiments have been performed on the $\text{BERT}_{\text{LARGE}}$ victim model employing the first 1000 validation examples from SQuAD dataset. The original F1 score and EM measure of the victim model, calculated before carrying out the attack algorithms, are 72.3% and 57.1%, respectively. The victim model's performance results are summarized in Table F.1.

Furthermore, we have expanded the training dataset of SQuAD by combining an additional 10% of adversarial examples generated through the BertAttack algorithm using the RobustQA framework. The augmented training dataset is used in training the $\text{BERT}_{\text{LARGE}}$ model. Finally, to demonstrate the effect of adversarial training on QA models, we have evaluated this model on six different adversarial attack algorithms using RobustQA. The results of the evaluation on 1000 validation examples of from the SQuAD dataset is shown in Table F.2.

G Experiment Setup

The computational experiments in this study were conducted on a system with an Intel Core i7-8700K CPU 3.70GHz 6-Core, a GeForce GTX 1080 8GB vRAM, and 64GB of RAM.

```
from qa.victim.question_answering.transformers import TransformersQuestionAnswering
from qa.attackers.textfooler import TextFoolerAttacker
from qa.metric import QAScore, EditDistance
from qa.attack_eval import AttackEval
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from datasets import load_dataset

model_name = "bert-large-uncased-whole-word-masking-finetuned-squad"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
victim = TransformersQuestionAnswering(
  model=model,
  tokenizer=tokenizer,
  embedding_layer=model.bert.embeddings.word_embeddings,
  device="cuda",
  max_length=512,
  truncation=True,
  padding=True,
  batch_size=8,
  lang="english"
)
dataset = load_dataset("squad", split="validation[0:1000]")
attacker = TextFoolerAttacker(tokenizer=victim.tokenizer, max_length=512)
metrics = [
  QAScore(victim, "f1", "orig"),
  QAScore(victim, "f1", "adv"),
  QAScore(victim, "em", "orig"),
  QAScore(victim, "em", "adv"),
  EditDistance()
]
evaluator = AttackEval(attacker, victim, metrics=metrics)
evaluator.eval(dataset, visualize=True, progress_bar=True)
```

Figure D.3: Executing the TextFooler attack on $\text{BERT}_{\text{LARGE}}$ via Python code.

	Question
Original	What part of Luther's career was one of his most productive?
TextFooler	what **portion** of luther's **calling** was one of his most productive?
Original	What high maintenance part did Tesla's AC motor not require?
Sememe PSO	what **in** maintenance **percentage** did tesla's ac motor not **call**?
Original	Who was the main performer at this year's halftime show?
PWWS	who was the **principal** performer at this year's halftime show?

Table E.1: Adversarial examples of three attack algorithms are showcased. The TextFooler, Sememe PSO, and PWWS algorithms are executed on the original question of the given instances from the validation set of the SQuAD dataset. The modified segments influenced by each attack are highlighted using bold text.

	TextFooler	PWWS	Genetic	Sememe PSO	BertAttack	DeepWordBug
Successful Instances	230	377	710	639	794	484
Attack Success Rate	0.23	0.37	0.71	0.63	0.79	0.48
Running Time	0.30	0.04	6.78	0.38	0.38	0.02
Victim Model Queries	85.81	61.33	1714.90	106.26	91.18	15.16
Exact Match (Original)	57.1	57.1	57.1	57.1	57.1	57.1
Exact Match (Adversary)	46.2	39.4	17.7	23.5	12.5	31.6
F1 Score (Original)	72.3	72.3	72.3	72.3	72.3	72.3
F1 Score (Adversary)	59.1	50.8	23.2	30.3	17.2	40.7
Levenshtein Edit Distance	1.66	1.93	2.37	1.98	2.19	4.09
Fluency	1765.1	2101.3	2250.3	1719.6	941.7	1274.6
Word Modification Rate	0.15	0.16	0.22	0.19	0.32	0.37
Semantic Similarity	0.84	0.79	0.76	0.79	0.83	0.55
Jaccard Char Similarity	0.91	0.90	0.89	0.89	0.89	0.80
Jaccard Word Similarity	0.46	0.43	0.41	0.43	0.44	0.34

Table F.1: QA adversarial attacks evaluation on $\text{BERT}_{\text{LARGE}}$ using RobustQA.

	TextFooler	PWWS	Genetic	Sememe PSO	BertAttack	DeepWordBug
Successful Instances	263	394	667	620	746	566
Attack Success Rate	0.26	0.39	0.66	0.62	0.74	0.56
Running Time	0.29	0.04	7.51	0.37	0.39	0.02
Victim Model Queries	83.56	61.19	1845.8	104.49	94.56	15.24
Exact Match (Original)	47.8	47.8	47.8	47.8	47.8	47.8
Exact Match (Adversary)	38.7	34.8	17.0	21.9	13.5	24.0
F1 Score (Original)	64.2	64.2	64.2	64.2	64.2	64.2
F1 Score (Adversary)	52.0	47.1	23.7	29.9	19.7	31.9
Levenshtein Edit Distance	1.54	1.85	2.37	1.90	2.08	4.62
Fluency	1130.7	1879.5	1642.2	1445.1	1001.3	1174.8
Word Modification Rate	0.15	0.15	0.23	0.19	0.31	0.42
Semantic Similarity	0.86	0.80	0.77	0.80	0.83	0.54
Jaccard Char Similarity	0.92	0.91	0.89	0.90	0.90	0.79
Jaccard Word Similarity	0.46	0.45	0.42	0.44	0.45	0.31

Table F.2: QA adversarial training evaluation on $\text{BERT}_{\text{LARGE}}$ trained on SQuAD and 10% of adversarial examples generated by the BertAttack algorithm in RobustQA.