Unicode Evil: Evading NLP Systems Using Visual Similarities of Text Characters

Antreas Dionysiou
University of Cyprus
Nicosia, Cyprus
adiony01@cs.ucy.ac.cy

Elias Athanasopoulos
University of Cyprus
Nicosia, Cyprus
eliasathan@cs.ucy.ac.cy

ABSTRACT

Adversarial Text Generation Frameworks (ATGFs) aim at causing a Natural Language Processing (NLP) machine to misbehave, i.e., misclassify a given input. In this paper, we propose EvilText, a general ATGF that successfully evades some of the most popular NLP machines by (efficiently) perturbing a given legitimate text, preserving at the same time the original text's semantics as well as human readability. Perturbations are based on visually similar classes of characters appearing in the unicode set. EvilText can be utilized from NLP services' operators for evaluating their systems *security* and *robustness*. Furthermore, EvilText outperforms the state-of-the-art ATGFs, in terms of: (a) *effectiveness*, (b) *efficiency* and (c) *original text's semantics and human readability preservation*. We evaluate EvilText on some of the most popular NLP systems used for sentiment analysis and toxic content detection. We further expand on the generality and transferability of our ATGF, while also exploring possible countermeasures for defending against our attacks. Surprisingly, naive defence mechanisms fail to mitigate our attacks; the only promising one being the restriction of unicode characters use. However, we argue that restricting the use of unicode characters imposes a significant trade-off between security and usability as almost all websites are heavily based on unicode support.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → *Machine learning*;

KEYWORDS

adversarial machine learning; adversarial text generation; natural language processing

ACM Reference Format:
Antreas Dionysiou and Elias Athanasopoulos. 2021. Unicode Evil: Evading NLP Systems Using Visual Similarities of Text Characters. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security (AISec '21), November 15, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3474369.3486871

AISec '21, November 15, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8657-9/21/11…$15.00
https://doi.org/10.1145/3474369.3486871

1 INTRODUCTION

A large number of Internet services and applications are heavily based on Natural Language Processing (NLP) techniques for processing, interpretation and manipulation of text. The advances of Deep Learning (DL) based NLP technologies have led to a broad deployment of such systems on important real-world problems. These DL-based NLP systems are often used for text classification problems, such as sentiment analysis and toxic content detection, demonstrating high accuracy rates. Nonetheless, these models have been shown to be vulnerable against sophisticated adversarial samples that are generated through the *perturbation*[1] of legitimate ones [16, 30, 31].

The nature of input data, which lacks of stationary distribution, makes Machine Learning (ML) algorithms vulnerable to different types of attacks, such as (a) poisoning the training data, e.g., crafting malicious input data that, when retrained on, causes the learner to learn an incorrect decision-making function, and (b) misleading the learner's classifications, essentially in a highly targeted manner, e.g., perturb a toxic comment that will be classified by the learner as non-toxic or the opposite. As a result, a new research topic came up, namely *adversarial ML*, which is mainly concerned about the design of ML algorithms that can resist these sophisticated attacks, as well as the study of the attackers' capabilities and limitations [5, 22, 44]. In this paper, we focus on (b); we carefully examine the potentials and capabilities of the attackers on misleading the learner's classification decisions through the perturbation of legitimate input samples [16, 30, 31].

Adversarial text generation, i.e., the generation of malicious text samples by perturbing legitimate ones, is a challenging task compared, for instance, to adversarial image generation, mainly because of the text's discrete nature, which makes attacks' optimization rather difficult [30, 31]. Additionally, text's semantics are fairly sensitive to deliberate perturbations/modifications [30]. Adversarial Text Generation Frameworks (ATGFs) have a wide-range of applications, and most notably in systems that need to filter-out malicious text, e.g., social media platforms and movies/TV shows reviewing sites. Many previous works focus solely on the image domain [2, 49], where it is relatively *easier* to construct malicious input samples which are virtually imperceptible to human perception [30]. In contrast, even small perturbations to text samples will be clearly perceptible to human eye although they may affect the classification outcome of a DL-based NLP system. In general, the image related known attacks cannot be directly utilized for the adversarial

[1] By saying *perturbation* we mean the transformation of a legitimate text to a malicious one by replacing specific words or characters. For example, transforming the word "bad" to "bâd" converts the sentiment detected by Google NLP platform from negative to positive.

text generation field. Thus, new attack/perturbation techniques (and the corresponding defences) are required, for causing NLP systems to misclassify a given input sample, being at the same time painless for humans to classify the same sample correctly [30].

Due to the increasing interest on adversarial text generation topic, some recent proposals came out suggesting numerous ways of generating adversarial text samples through the perturbation of legitimate ones [16, 30, 31, 43]. Nonetheless, all the aforementioned studies are conducted under specific settings and assumptions affecting either their *generality and applicability* or their *effectiveness and efficiency*. Thus, the performance, in terms of effectiveness, efficiency and practicality, of popular ATGFs has to be carefully assessed. Some examples of assumptions and limitations affecting some of the most popular ATGFs are shown in Table 1.

In this paper, we propose EvilText[2], a *black-box* based ATGF specifically designed to effectively and efficiently perturb a given legitimate text for causing popular NLP models to misbehave, preserving at the same time the original meaning for human readers. EvilText perturbs text by leveraging visual similarities of characters included in the *unicode set*. In particular, EvilText offers a series of perturbation tactics for crafting malicious text samples by replacing the legitimate text's characters with other, visually similar ones, from the unicode set. EvilText can be utilized from different, DL-based or not, NLP services' administrators for evaluating their system's security and robustness. Hence, EvilText is *not* just another ATGF, rather than also an effective *evaluation methodology* for the actual security of current NLP systems. Although different perturbation tactics, each one having different performance, exist; EvilText offers attacking methodologies that achieve high attack success rates, while also preserving at the most the human readability factor. Thus, this paper also makes a contribution by exploring a specific range of all the possible attacks spectrum of adversarial text generation techniques. This will help the scientific community to develop generally applicable defences as the current defence strategies cannot be trivially deployed while also imposing significant trade-offs.

Our attacks are *black-box* based, i.e., agnostic about the target model's internals, thus, minimizing the adversarial assumptions compared to the white-box setting [16]. In addition, we demonstrate the *transferability*, i.e., general applicability, of our attacks by evaluating EvilText's performance on some of the most popular black-box and white-box NLP machines (we choose NLP machines that cause other state-of-the-art ATGFs to achieve the *lowest* attack success rates on average), *outperforming*, in almost all cases, the state-of-the-art ATGFs. Note that we treat both white-box and black-box target NLP machines in a *black-box manner*, being able to only query each target NLP model and receive its prediction as a response. In particular, EvilText manages to evade NLP systems without even requiring either the numerical confidence score or the output label from the target model, in contrast to other approaches found in the literature, e.g., [16, 30]. Furthermore, we are the *first* to report attack success rates on *both*: (a) negative sentiment/toxic sample to positive sentiment/non-toxic sample, and (b) positive sentiment/non-toxic sample to negative sentiment/toxic sample, types of attack, in contrast to all other approaches that either focus *solely* on the (a) type of attack [16, 17, 30] or alter the original text's semantics [31]. Moreover, we are the *first*, at least to our knowledge, to report results on the MR_v2.0 dataset [33].

Finally, we explore potential defence strategies and explain the reason why naive defences, such as spell checking and adversarial training, are *not* effective for mitigating our attacks. In fact, we suggest that one practical countermeasure for defending against EvilText is the restriction of unicode characters use. However, this is a highly unrealistic scenario as almost all of the websites are heavily based on unicode support, i.e., a trade-off between security and usability exists.

Our contributions can be summarized as follows.

- We deliver a new *unicode map* containing the visually similar to English alphabet letters unicode characters. Our unicode map is of larger length compared to other similar maps, e.g., [15], and thus, of higher *utility*, meaning that it offers more perturbation options for each letter.
- We propose EvilText, an ATGF that is able to conduct successful and easy to implement attacks, using the adversarial text produced by perturbing the legitimate text, while preserving at the same time the text's original semantics and human readability. For instance, EvilText achieves 100% attack success rate on MR_v1.0, MR_v2.0 and IMDB datasets when targeting the Logistic Regression (LR) model. Moreover, EvilText can be utilized from different NLP services' operators for evaluating their system's *security* and *robustness*. Finally, EvilText can be extended to include other TPTs that might evade specific target NLP machines.
- We evaluate EvilText on some of the most popular ML models and real-world online NLP applications, including sentiment analysis and toxic content detection, *outperforming* the state-of-the-art ATGFs in almost all cases, while also being the *first*, at least to our knowledge, to report results on the MR_v2.0 dataset [33].
- We, for the first time, demonstrate that the positive sentiment/non-toxic sample to negative sentiment/toxic sample type of attack is *more difficult* to be performed than the opposite one.
- We conduct a user study on our generated adversarial texts showing that EvilText has *little* impact on human understanding and original text's semantics.
- We explore potential defence strategies for mitigating our attacks. Our results suggest that *naive* defences, such as spell checking and adversarial training, cannot resist against our attacks. In fact, one practical countermeasure might be the restriction of unicode characters use which, however, imposes a *trade-off* between security and usability as almost all of the websites are heavily based on unicode support.

2 EVILTEXT

As shown in Figure 1, our framework is built upon the following 4 components: (a) the positive/negative opinion words lists, (b) the perturbation/attack tactics, (c) the *unicode map* containing visually similar to English alphabet letters unicode characters, and (d) the similar positive/negative opinion words dictionary. EvilText's attack pipeline is straightforward. First, it receives as input, the legitimate text along with the attack/perturbation method selected

[2]EvilText is freely available on Bitbucket (https://bitbucket.org/srecgrp/eviltext/).

Table 1: An overview of the settings and assumptions affecting the effectiveness/efficiency or the generality/applicability of popular ATGFs found in the related literature.

Assumptions/Limitations	Reference papers	Effectiveness /Efficiency	Generality /Applicability
1. White-box (and thus not realistic) attacks only	[7, 18, 37, 45]		•
2. Attacks oriented only a small group of NLP machines	[16]		•
3. Negative sentiment/toxic sample to positive sentiment/non-toxic sample attack but not the opposite	[16, 17, 30, 31]		•
4. Negative impact on the original text's semantics	[31]	•	
5. Not as effective/efficient as our framework in black-box attack setting	[30]	•	
6. Attacks requiring heavy manual intervention	[3, 27]	•	•
7. Attacks requiring the target model's confidence score or its output label	[16, 17, 30, 31]	•	•

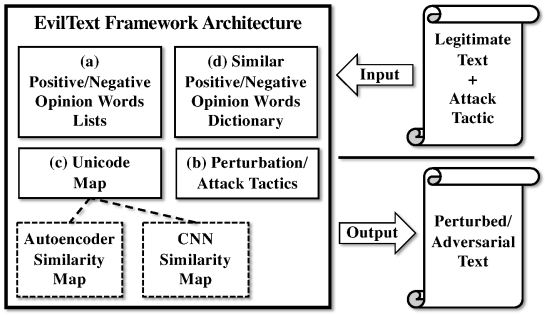


Figure 1: EvilText consists of 4 components: (a) the positive/negative opinion words list (drawn from [21]), (b) the perturbation/attack tactics, (c) the unicode map, and (d) the similar positive/negative opinion words dictionary. EvilText takes as input legitimate text along with the selected attack tactic and responds with the adversarial text.

and, second, it responds with the perturbed/adversarial text. After that, the attacker submits the adversarial text to the target NLP machine. The simplistic design as well as the *standalone* nature of EvilText framework facilitates its use across a wide-range of different NLP applications and further strengthens the *transferability property* of our attacks. Thus, the different NLP services' operators are able to evaluate their systems' security and robustness without significant effort. Note that we explore various Text Perturbation Tactics (TPTs) from naive to more intelligent ones (see Section 3).

Unicode Map Construction. Our attacks are heavily based on replacing the existing alphabet characters, contained in the given input text, with other visually similar ones from the unicode space. Thus, we came up constructing a *unicode map* containing the visually similar unicode characters for each English alphabet letter (26 in total). In order to construct our unicode map, which will be used for crafting the adversarial texts by perturbing the given/legitimate ones, we utilize two DL techniques, namely: (a) Convolutional Neural Networks (CNNs) [28], and (b) Autoencoders [42]. Although there are other possible options for constructing this map we choose to use ML-based techniques because they fall into our field of expertise while also achieving state-of-the-art performance on image classification/recognition tasks [11, 12, 26]. The final unicode map is the result of taking the *union* of the two maps derived from the two aforementioned DL techniques. Using both CNNs and Autoencoders, for finding visually similar characters from the unicode space, will result in a unicode map of higher quality, compared to the one created using one technique only. This is because CNNs and Autoencoders may detect different unicode characters as similar to specific alphabet letters. Thus, using both ANNs helps us to decide about the most visually similar ones by utilizing the confidence score of each classification. In addition, this *hybrid* DL-based method used for grouping unicode's visually similar characters, can also be easily extended to other visual similarity classification tasks. A snippet of the final unicode map is shown in Figure 2. Note that during the adversarial text generation process, each unicode character contained in the final unicode map will be chosen randomly, i.e., using a uniform distribution probability.

For the CNN we use the well-known Lenet-5 architecture [29] trained on the EMNIST-letters dataset [9] which contains 145, 600 handwritten characters in a total of 26 balanced classes. After training our CNN classifier and achieving *99.20% testing accuracy*, we have provided as input the image representations of all unicode characters. Next, we have created a list containing the most (visually) similar characters found in the unicode space, for each English alphabet letter, in decreasing order of similarity (according to CNN's confidence score). Note that we set a threshold to the classifier's confidence (≥ 0.95) for including only unicode characters with significant visual similarity. Our convolutional Autoencoder [19] is composed of two sub-networks, the encoder and the decoder. The encoder sub-network compresses the input image sample (matches the EMNIST-letters dataset format) to a 32-dimensional (32D) vector, i.e., the code, and the decoder sub-network reconstructs the initial input image sample from the 32D vector created by the encoder sub-network, with the minimum possible reconstruction loss. We detect the most similar unicode characters by calculating the Euclidean distances between the reduced encoding of the given input and all the other learned encodings, sorting them in increasing order. Similar to our CNN classifier, we set a threshold to the calculated Euclidean distances (≤ 0.1) for selecting only unicode characters with significant visual similarity.

Our unicode map is more practical compared to other similarity maps, such as [15], as it creates 26 clusters (one for each English alphabet letter) grouping all unicode characters (capitals or lowercase) into the closest cluster. This results to a list of larger length and thus of higher *utility* [3], compared to [15]. According to [15], their image-processing based method (which matches similar characters based only on specific points on the characters' contour) *does not* perform well when certain amount of shift of the glyph contour

[3]By saying *utility* we mean the number of possible options for replacing one character with another, which is, however, visually similar.

Figure 2: A snippet of the *unicode map* with the visually similar unicode characters. Note that some characters look identical, but they are essentially different in the unicode space.

exists. Therefore, we instead deploy DL-based techniques, i.e., CNNs and Autoencoders, as they are specifically designed to ensure some degree of shift, scale, and distortion invariance [28].

Positive/Negative Opinion Words Lists. Performing targeted perturbation of specific positive/negative sentiment or toxic/non-toxic words can substantially improve the performance of any proposed ATGF, in terms of *effectiveness*, i.e., higher attack success rates, *efficiency*, i.e., faster adversarial text generation process, and *original text's semantics preservation*. Thus, we utilize the two lists of positive and negative opinion words derived from [21], where the authors study the problem of generating feature-based summaries of customer reviews of products sold online. In the majority of our proposed TPTs, we deliberately perturb the words contained in the *intersection* of the given legitimate text, i.e., the text to be perturbed, and the two positive/negative sentiment words lists drawn from [21]. As a result, our attacks minimize the computational cost required for crafting adversarial texts by performing targeted perturbations of the most sentiment/toxicity defining words.

Similar Opinion Words Dictionary. Some positive or negative sentiment words are of small length. This fact drastically limits the number of possible perturbation combinations that could be performed benefiting the NLP systems at detecting the actual sentiment/toxicity of a given text. In order to tackle this problem, we have created a list with the similarity scores of positive and negative sentiment words, utilizing the Global Vectors for Word Representation (GloVe) [40] model, for replacing small length words with semantically similar and larger ones. In particular, we use GloVe's 50-dimensional (50D) pre-trained word vectors derived from Wikipedia 2014 and Gigaword 5. After collecting and storing the 50D vector for each positive and negative sentiment word found in the two words lists derived from [21], we calculate the Euclidean distances between them in order to conclude about the most similar ones and thus create the *similar positive/negative opinion words dictionary*. Finally, for successfully conducting the third attack, i.e., TPT 3 described in Section 3, small length positive/negative sentiment and toxic/non-toxic words will be replaced with other words of similar sentiment or toxicity and of larger length, from the similar positive/negative opinion words dictionary. We set a threshold for selecting the top-5 similar words, i.e., the 5 words having the minimum Euclidean distance with the small length to-be-replaced word, so that the replaceable words are guaranteed to be semantically similar to the original ones.

3 ATTACKS

Attacks' Design. A large number of possible TPTs exists. However, in this paper, we focus on a subset of all possible TPTs and especially the ones that *minimize* the impact on the semantics or the original meaning of the initial input text. DL-based NLP machines have a pre-defined *unknown word vector* for out-of-vocabulary (OOV) words. Thus, we deliberately target specific words that demonstrate sentiment or toxicity in order for those words to be mapped to the unknown word vector. Our results suggest that this strategy causes all tested DL-based NLP machines to misbehave, when trying to classify the given text's sentiment or toxicity. Furthermore, our attacks are partly inspired from the fact that *randomly* changing words would not fool classifiers. Thus, choosing *important words* to modify is necessary for successful attacks [30]. We propose both *undirected* and *directed* TPTs in the sense that we either perturb the whole input text given (undirected) or specific words that largely affect the target NLP model's classification outcome (directed).

Threat Model. The attacker's goal is to cause the target ML model to perform a misclassification by perturbing the initial text, without, however, changing its original semantics for human readers. An attacker can do so, either using attacks in white-box or black-box setting.

White-box attack setting assumes that the user can infer the inner structure, e.g., the gradients, of a particular ML model, and thus has a complete knowledge of the target model.

Black-box attack setting assumes that the user is not aware of the model's architecture, parameters, or training data, being only capable of querying the target model and receiving its predictions/confidence scores for each class as a response. Furthermore, the fact that such platforms allow for a limited number of free requests has to be taken into explicit consideration by potential attackers that want to perform practical attacks [30].

In this paper, we focus solely on conducing attacks in the *black-box* setting due to its inherent practicality. In other words, we treat even white-box models, for which we know their internal structure, in a black-box manner, i.e., we can only query them and receive their confidence score vector as a response. Conducting such attacks in black-box setting is arguably the most common case as in most real-world scenarios a user cannot examine or retrieve/download the inner structure of a target ML model [16].

Text Perturbation Tactics (TPTs). After experimenting with various NLP machines and considering different ways of modifying a given input text, but at the same time preserve its original meaning, we came up with a list of possible TPTs. However, EvilText allows any potential attacker to *extend* those TPTs and include new ones that specifically affect certain NLP systems. In fact, this is the main reason why we choose *not* to combine our TPTs into a single TPT as different NLP machines are affected by different TPTs. As a result, adversaries can utilize a single or a combination of TPTs that best fit their needs, i.e., they achieve the highest evasion success rates when targeting specific NLP services.

Similar to [30], we offer TPTs targeting specific sentiment/toxicity defining words. However, in [30]'s case, for determining those words, the authors iteratively query the target model by excluding one word at a time, which decreases their attacks' efficiency. In contrast, we directly perturb the words included in the intersection of the given text and the positive/negative opinion words lists derived from [21]. As a result, EvilText is *substantially more efficient* compared to TextBugger as it eliminates the computation (and communication) overhead required for determining those (already

Figure 3: *EvilText* pipeline: the attacker (1) feeds the target NLP machine with the legitimate text and receives back a report, (2) gives as input the legitimate text along with the selected attacking method and *EvilText* responds with the adversarial text, and (3) feeds the adversarial text to the target NLP machine and the machine responds with a false report. This 3-step process is applied for evading both sentiment analysis and toxic content detection machines.

detected and recorded by the literature) words. In addition, our unicode map is of larger length compared to other similar maps, e.g., [15], thus, offering a wider-range of choices for characters' perturbation. A complete list containing examples for all EvilText's TPTs is shown in Table 2.

Table 2: Examples for all adversarial TPTs.

Original Text: —————————————— ...post-traumatic stress disorder is kinda dumb.
TPT 1 (The Naive Attacking Method) —————————————— ...þŎ5Ŧ-TŔȺʋɱÂʅƚc ŠŤȑĘƧ5 dİşŏȒďèŘ íš ĸɨñďä ɗuMþ.
TPT 2 (Targeted Perturbation of Negative/Positive Sentiment Words) —————————————— ...post-traumatic ƽtręƧş dİšòŕĐÈȒ is kinda dumb.
TPT 3 (Replacing Small Length Negative/Positive Sentiment Words) —————————————— ...post-traumatic ȂŋϰȉÊŁŶ şȖʄƒEɼƖηĜ is kinda dumb.
TPT 4 (Doubling Negative/Positive Sentiment Words) —————————————— ...post-traumatic stressstress disorderdisorder is kinda dumb.
TPT 5 (Doubling Negative/Positive Sentiment Words and Perturb) —————————————— ...post-traumatic ŠŢŖęŚşŠţȒEşş ĐĬšóŔɖĖɼĐĭșóŔđÊɼ is kinda dumb.
TPT 6 (Insert Spaces Between Negative/Positive Sentiment Words' Letters) —————————————— ...post-traumatic s t r e s s d i s o r d e r is kinda dumb.
TPT 7 (Insert Spaces Between Negative/Positive Sentiment Words' Letters and Perturb) —————————————— ...post-traumatic Ŝ ł ғ E š Ŝ Ď Í ƽ Ő ɼ Ɖ Ȅ Ř is kinda dumb.

TPT 1: The Naive Attacking Method. The naive attacking method replaces all English alphabet letters found in the given legitimate text with visually similar characters derived from the unicode map. If a non English alphabet character is found, it is kept the same in the new perturbed text. As a result, the new perturbed text is of the same length as the original one.

TPT 2: Targeted Perturbation of Negative/Positive Sentiment Words. This TPT only perturbs the negative/positive sentiment words contained in the intersection of the given legitimate text and the two negative/positive sentiment words lists derived from [21], using our unicode map (Section 2).

TPT 3: Replacing Small Length Negative/Positive Sentiment Words. After constructing the two lists containing the top-5 similar positive/negative sentiment words for each word found in both positive/negative sentiment words lists derived from [21], we replace each occurrence of positive/negative sentiment word found in the given input text with a randomly chosen, and of bigger length word from the top-5 similar positive/negative sentiment word lists. Next, we perturb the new words, which are however semantically similar to the previous ones, using our unicode map.

TPT 4: Doubling Negative/Positive Sentiment Words. This TPT doubles the negative/positive sentiment words found in both the given input text and the negative/positive sentiment words lists derived from [21]. In particular, for each negative/positive sentiment word found, we copy the same word and output it twice, one next to the other. For example, if the word *"bad"* is found, we output the transformed word *"badbad"* in the same position that the previous word existed.

TPT 5: Doubling Negative/Positive Sentiment Words and Perturb. For this TPT we perform the same procedure as the previous one, i.e., TPT 4, but we also perturb the transformed word utilizing the unicode map. Each letter of the two (same) words is randomly perturbed, i.e., for each letter a visually similar unicode character is randomly chosen from the unicode map.

TPT 6: Insert Spaces Between Negative/Positive Sentiment Words' Letters. This TPT inserts spaces between each character of the negative/positive sentiment words found in both the given input text and the negative/positive sentiment words lists derived from [21]. For example, if the word *"bad"* is found, we output the transformed word *"b a d"* in the same position that the previous one existed.

TPT 7: Insert Spaces Between Negative/Positive Sentiment Words' Letters and Perturb. For this TPT we perform the same procedure as the previous one, i.e., TPT 6, but we also perturb the transformed word, i.e., the negative/positive sentiment word with spaces between each character, utilizing the unicode map.

4 EVILTEXT EVALUATION

We evaluate our attacks on a number of different black-box/white-box sentiment analysis as well as toxic content detection NLP machines. Furthermore, in order to have a direct comparison with the state-of-the-art ATGFs [16, 17, 30], we choose to evaluate our attacks on a subset of the machines tested in the aforementioned studies. In order to avoid any bias regarding the experimental datasets as well as to demonstrate the general applicability/transferability of our framework, we evaluate EvilText on two different NLP problems, namely: (a) *sentiment analysis* and (b) *toxic content detection.*

The two attack pipelines are shown in Figure 3. Our evaluation setup follows that of prior work in the field such as [16, 17, 30]. We do not provide the average time needed for generating adversarial samples for each dataset since EvilText requires 1 pass of the legitimate text to complete the attack process (no iterative optimization is involved). For instance, EvilText perturbs the whole MR_v1.0 dataset in just *2.06 seconds* on a 4-core Xeon machine. Thus, we avoid providing a detailed analysis and comparison of EvilText's efficiency with other state-of-the-art ATGFs since our framework is faster due to its standalone nature [4]. Instead, in this section, we focus on the effectiveness, i.e., success rates of evading NLP systems, and semantics preserving aspects.

Sentiment Analysis. Sentiment analysis refers to the use of NLP techniques, such as statistics and/or ML methods, to extract, identify, or characterize the sentiment of a text unit [30]. Sentiment analysis has been successfully applied to a wide-range of applications, such as opinion mining [35], especially for improving businesses' communications with their clients. We evaluate EvilText's performance on a number of popular DL-based black-box, i.e., online NLP plaforms, and white-box sentiment analysis machines. In order to have a direct comparison with state-of-the-art ATGFs we choose to evaluate our attacks on a subset of the sentiment machines used in [16, 17, 30].

In total, we target 3 *white-box* sentiment analysis machines, namely LR, Kim's CNN [25] and the Long Short-term Memory (LSTM) (used in [47]). The classifiers' hyper-parameters are fine-tuned according to the sensitivity analysis on model performance conducted by Zhang et al. [48] (as done in [30]). Furthermore, we use the same hold-out test strategy, i.e., 80%, 10% and 10% of the data was used for training, validation and testing, respectively, while also tuning the hyper-parameters only on the validation set, exactly as done in TextBugger [30].

In addition, we target 3 *black-box* sentiment analysis machines, namely FastText, TextProcessing and Aylien. However, we focus on collecting the results from FastText machine as it has been experimentally shown that FastText performs best, i.e., it is the most robust classifier allowing the *lowest* attack success rates on average, compared to the other black-box NLP machines [30] [5]. One potential explanation about FastText's robustness against such attacks is its subword based embedding nature (used from state-of-the-art NLP models) [13]. The FastText model is trained on the Amazon Review Polarity [24] dataset and we do not have any information about the model's parameters or architecture.

Datasets. We evaluate EvilText's performance on the following 3 benchmark datasets for sentiment analysis.

- **IMDB [32]:** This dataset contains 50,000 positive and negative movie reviews crawled from online sources, having an average length of *216* words per sample. It has been divided into two parts, i.e., 25,000 reviews for training and 25,000 reviews for testing.
- **Rotten Tomatoes Movie Reviews Version 1 (MR_v1.0) [34]:** This dataset is a collection of movie reviews collected by Pang and Lee [34]. It contains 5,331 positive and 5,331 negative processed sentences/snippets, having an average length of *32* words per sample.
- **Rotten Tomatoes Movie Reviews Version 2 (MR_v2.0) [33]:** This dataset represents an enhancement of the MR_v1.0 dataset. It contains 1,000 positive and 1,000 negative processed reviews with multiple lines and average length of *706* words per sample. We are the *first* to report results on this dataset, at least to our knowledge.

[4]For instance, some preliminary results showed that EvilText is ≈ 10× more efficient compared to TextBugger [30], when tested on MR_v1.0 dataset.
[5]Our evaluation results suggest the same.

Table 3: The sentiment analysis target models' testing accuracies used in TextBugger and EvilText ATGFs.

Model	Dataset	TextBugger [30]	EvilText
LR	MRv1	73.70%	**75.43%**
	MRv2	–	**79.32%**
	IMDB	82.1%	**88.04%**
CNN	MRv1	78.10%	**80.64%**
	MRv2	–	**78.15%**
	IMDB	89.4%	**89.9%**
LSTM	MRv1	80.10%	**84.41%**
	MRv2	–	**86.50%**
	IMDB	90.70%	**93.20%**

Attacks' Effectiveness and Efficiency. The *testing* accuracies for the 3 white-box sentiment analysis machines used, i.e., LR, CNN and LSTM, are shown in Table 3. As shown, EvilText targets sentiment analysis machines with higher performance compared to state-of-the-art ATGF, namely TextBugger [30]. This means that our white-box machines are essentially *better* at detecting the actual sentiment contained in a given text, compared to [30]. This fact makes it *harder* for our ATGF to evade the tested, DL-based or not, sentiment analysis machines. However, our results indicate that all the TPTs proposed, as part of EvilText ATGF, successfully evade some of the most popular NLP sentiment analysis machines.

Table 4 shows the attack success rates for negative and positive reviews, on the IMDB, MR_v1.0 and MR_v2.0 datasets, for all TPTs. As shown, the LR classifier is more susceptible to *negative* adversarial texts compared to the other two DL-based NLP models, i.e., CNN and LSTM. However, the LR classifier is extremely robust on classifying *positive* sentiment adversarial reviews as we manage to achieve no more than *25.05%* attack success rate, whereas for the other two DL-based NLP models we achieve 56.04% and 88.94% attack success rates, respectively. Actually, it is rather interesting that the LSTM model, which achieves higher testing accuracy compared to LR and CNN (see Table 3), performs worse on classifying positive sentiment adversarial reviews.

In Figure 4, we compare EvilText's performance on evading white-box and black-box NLP sentiment analysis machines, against state-of-the-art ATGFs. Note that we report attack success rates on *both* negative and *positive* reviews in contrast to state-of-the-art ATGFs which focus *solely* on attacking negative reviews. As shown in Figures 4(a)-4(c), our TPTs *outperform*, in almost *all* cases, the state-of-the-art white-box ATGFs. The only case where TextBugger [30] achieves better attack success rates, is on the IMDB dataset tested on the white-box CNN machine having, however, little improvement, i.e., only 0.88%, compared to our ATGF. Moreover, this result can be explained by the fact that our white-box classifiers are more robust on detecting the actual sentiment contained in a given

Table 4: The attack success rates for each TPT on the negative (upper row) and positive (lower row) sentiment reviews contained in each dataset, for all white-box sentiment analysis machines.

Model	Dataset	TPT 1	TPT 2	TPT 3	TPT 4	TPT 5	TPT 6	TPT 7
LR	MR_v1.0	**100%**	86.73%	86.73%	86.73%	86.73%	86.56%	86.56%
		0%	5.87%	5.87%	5.87%	5.87%	**6.04%**	**6.04%**
	MR_v2.0	**100%**	74.60%	74.60%	74.60%	74.60%	74.70%	74.70%
		0%	9.5%	9.5%	9.5%	9.5%	**9.6%**	**9.6%**
	IMDB	**100%**	38.63%	38.64%	38.64%	38.64%	38.52%	38.52%
		0%	24.96%	24.96%	24.96%	24.96%	**25.05%**	**25.05%**
CNN	MR_v1.0	**88.47%**	36.47%	36.44%	36.47%	36.47%	67.66%	52.19%
		8.88%	**26.59%**	**26.59%**	26.38%	**26.59%**	5.83%	14.31%
	MR_v2.0	**92.65%**	30.90%	30.90%	30.90%	30.90%	38.00%	37.50%
		4.40%	18.40%	18.40%	**18.60%**	18.40%	14.10%	13.40%
	IMDB	**89.62%**	28.14%	28.12%	28.12%	28.13%	38.37%	32.12%
		9.11%	55.79%	**56.04%**	54.96%	55.79%	41.83%	49.65%
LSTM	MR_v1.0	**86.46%**	48.15%	48.15%	48.44%	48.44%	50.75%	54.11%
		86.46%	48.15%	48.15%	48.44%	48.44%	50.75%	54.11%
	MR_v2.0	**84.03%**	19.90%	20.00%	19.90%	19.90%	25.40%	28.60%
		18.55%	26.10%	26.10%	25.50%	26.10%	**43.00%**	22.40%
	IMDB	**94.02%**	34.00%	34.01%	34.00%	34.01%	12.51%	36.22%
		88.94%	34.19%	34.20%	33.35%	34.20%	40.03%	35.09%

Table 5: The attack success rates for each TPT on the negative (upper row) and positive (lower row) sentiment reviews contained in each dataset, for all black-box sentiment analysis machines.

Model	Dataset	TPT 1	TPT 2	TPT 3	TPT 4	TPT 5	TPT 6	TPT 7
FastText	MR_v1.0	16.80%	30.20%	30.55%	30.65%	30.66%	**34.06%**	32.82%
		82.36%	39.24%	39.31%	39.37%	39.39%	35.97%	36.69%
	MR_v2.0	**96.50%**	48.30%	48.30%	49.40%	47.60%	54.60%	52.40%
		2.90%	27.30%	**27.90%**	27.20%	27.40%	19.20%	23.70%
	IMDB	**36.13%**	24.23%	24.31%	24.49%	24.26%	27.16%	26.21%
		61.97%	36.76%	36.64%	36.71%	36.92%	27.70%	32.74%
TextProcessing	MR_v2.0	24.40%	3.90%	**42.40%**	3.90%	3.90%	3.60%	3.10%
		63.10%	42.40%	3.90%	42.30%	42.40%	35.30%	38.40%
Aylien	MR_v2.0	**98.60%**	72.20%	72.30%	72.10%	71.80%	81.50%	75.30%
		1.70%	**41.70%**	41.60%	**41.70%**	**41.70%**	26.50%	37.80%

text as they achieve higher testing accuracies, compared to [30], thus, being more difficult to be fooled. Furthermore, in all cases, FGSM+NNS [17] and DeepFool+NNS [17] perform the worst. Finally, EvilText achieves 100%, 92.65% and 84.03% attack success rate on LR, CNN and LSTM sentiment analysis machines, respectively, on the MR_v2.0 dataset.

For the FastText black-box sentiment analysis machine (Figure 4(d)), EvilText achieves higher attack success rates, compared to the other state-of-the-art ATGFs, on the MR_v1.0 dataset, i.e., 16.86% and 45.36% higher attack success rate from TextBugger and DeepWordBug, respectively, but lower attack success rates on the IMDB dataset than those reported in TextBugger [30] and DeepWordBug [16]. In addition, EvilText achieves 96.50% attack success rate on the MR_v2.0 dataset. Notice, that in Figure 4, we report the highest attack success rates achieved on either negative or positive sentiment reviews as the security implications are essentially the same.

Toxic Content Detection. Toxic content detection machines apply ML, statistics and syntax rules for the detection of toxic-related or illegal language use, e.g., insults, harassment and racism. One example for the use of such NLP systems are the online communication networks where the moderators are responsible for ensuring the appropriateness of the conversation environment [30].

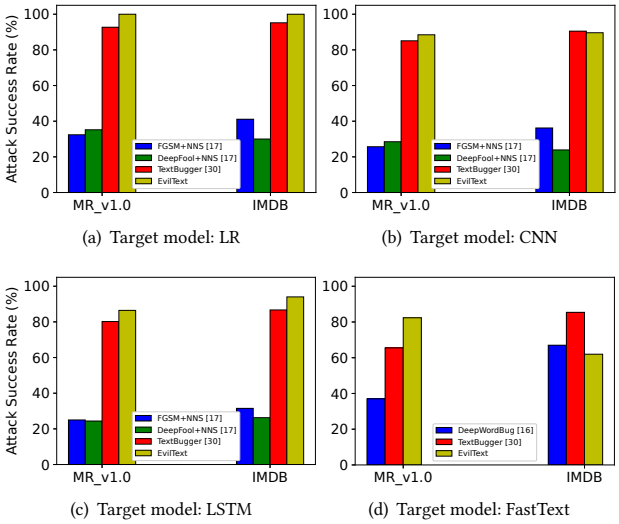
Table 6: The toxic content detection target models' testing accuracies used in TextBugger and EvilText ATGFs.

Model	TextBugger [30]	EvilText
LR	88.50%	**89.72%**
CNN	93.50%	**95.44%**
LSTM	90.70%	**91.89%**

In order to have a direct comparison with the state-of-the-art ATGFs, we choose the same white-box machines used in sentiment analysis section. However, for the attacks on black-box machines we focus solely on FastText as it has been proven the most robust classifier, allowing the lowest attack success rates on average [30].

Dataset. We evaluate EvilText's performance on the popular benchmark dataset provided by Kaggle's Toxic Comment Classification competition[6]. This dataset contains a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. There are six types of indicated toxicity, i.e., *toxic*, *severe toxic*, *obscene*, *threat*, *insult*, and *identity hate*, in the original dataset. We consider these categories as toxic and perform binary

[6]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.



(a) Target model: LR (b) Target model: CNN

(c) Target model: LSTM (d) Target model: FastText

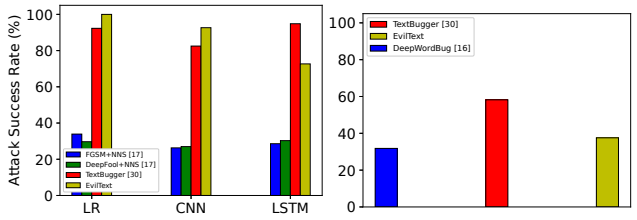
Figure 4: *Comparison* of state-of-the-art ATGFs' attack success rates on the 3 *white-box* 4(a)-4(c) and 1 *black-box* 4(d) sentiment analysis machines.

classification for toxic content detection on the balanced[7] dataset as Li et al. did in [30]. We do not perform any further filtering based on the samples' size (in terms of the total number of words) or characters contained, e.g., filtering out samples with repeated characters, as Li et al. [30] did. In total, we obtained *73,959* toxic and non-toxic samples, respectively, having an average length of 69 words per sample, in contrast to TextBugger [30] which considers *only 12,630* toxic and non-toxic samples, respectively. Thus, our attack success rates are reported on a *more representative* and *less biased* dataset compared to [30].

Attacks' Effectiveness and Efficiency. The *testing* accuracies for the 3 white-box toxic content detection machines used, i.e., LR, CNN and LSTM, are shown in Table 6. As shown, EvilText targets toxic content detection models with higher performance compared to state-of-the-art ATGF, namely TextBugger [30]. This means that our white-box machines are essentially *better* at detecting the actual toxicity in a given text, compared to [30]. This fact makes it *harder* for our ATGF to evade the tested, DL-based or not, toxic content detection machines. Nonetheless, our results indicate that all the TPTs proposed, as part of EvilText framework, successfully evade some of the most popular toxic content detection machines.

Table 7 shows the attack success rates for toxic and non-toxic samples, contained in the Kaggle's toxic comment classification competition dataset, for all TPTs. As shown, the LR classifier is more susceptible to *toxic* adversarial samples compared to the other two DL-based NLP models, i.e., CNN and LSTM. However, the LR classifier is extremely robust on classifying *non-toxic* adversarial samples as we manage to achieve no more than 1.92%, whereas for the other two DL-based models we achieve 6.91% and 36.52% attack success rates, respectively.

[7]We randomly select an equal number of non-toxic and toxic samples.



(a) White-box Machines (b) Black-box Machine (FastText)

Figure 5: Comparison of state-of-the-art ATGFs' attack success rates on 3 white-box (LR, CNN, LSTM – 5(a)) and 1 black-box (FastText – 5(b)) toxic content detection machine.

In Figure 5, we compare EvilText's performance on evading white-box and black-box toxic content detection machines, against the state-of-the-art ATGFs. Again, we report attack success rates on both toxic and *non-toxic* samples in contrast to other ATGFs which focus solely on attacking toxic samples. As shown, our TPTs are highly effective *outperforming*, in many cases, the current state-of-the-art ATGFs. Similar to the sentiment analysis scenario, FGSM+NNS [17] and DeepFool+NNS [17] achieve the lower attack success rates when targeting white-box toxic content detection machines. In addition, the only two cases where TextBugger [30] achieves better attack success rates is on the LSTM (white-box) and FastText (black-box) target models. However, these results can be explained for two reasons. First, for training our target models, we utilize a much larger dataset, i.e., *73,959* toxic and non-toxic samples instead of 12,630 toxic and non-toxic samples used in [30], as we do not perform the *strict filtering* [8] procedure described in [30] that may exclude a large number of legitimate samples. Second, as shown in Table 6, our trained white-box classifiers are more robust on detecting the actual toxicity in a given text as they achieve higher testing accuracies, compared to [30], making them more difficult to be fooled. Finally, EvilText reports *76.82%* attack success rate on the non-toxic to toxic sample type of attack, whereas the other state-of-the-art ATGFs *do not report any results at all.*

User Study. In order to conclude about whether or not our TPTs negatively affect either human understanding or the semantics of the initial text, we conduct a user study with (randomly selected) human participants. By doing so, we will observe whether or not the applied perturbations will change the human perception of the text's sentiment or toxicity. Our user study follows the same setup as [30]. After randomly choosing two adversarial texts crafted with each TPT, we compose a questionnaire with 14 questions in total, where the human participants are given evasion-successful adversarial texts, i.e., adversarial samples that have successfully fooled the target classifiers, and are asked to provide the initial (non-perturbed) text, in a given text-box, along with the difficulty level faced when interpreting each adversarial text, i.e., Likert scale questions with 5 points [6]. In this way, we are able to perform, not only a *quantitative* analysis, but also a *qualitative* analysis, which

[8]We advise researchers to perform the *least possible filtering* on the datasets, in order to: (a) avoid any bias, (b) retain the largest possible average word length, and (c) facilitate the accurate comparisons with other ATGFs.

Table 7: The attack success rates for each TPT on the toxic (upper row) and non-toxic (lower row) samples for all tested white-box (WB) and black-box (BB) toxic content detection machines.

Model	BB or WB	TPT 1	TPT 2	TPT 3	TPT 4	TPT 5	TPT 6	TPT 7
LR	WB	**99.99%**	94.76%	94.77%	94.76%	94.76%	94.40%	94.40%
		0%	1.88%	1.89%	1.89%	1.89%	**1.92%**	**1.92%**
CNN	WB	**99.69%**	91.68%	91.65%	88.51%	91.68%	88.29%	91.15%
		0.25%	4.13%	4.14%	4.12%	4.13%	**6.91%**	4.36%
LSTM	WB	60.56%	72.22%	72.21%	72.23%	72.22%	71.04%	**72.66%**
		36.52%	12.76%	12.75%	16.92%	12.75%	13.20%	12.15%
FastText	BB	21.11%	31.61%	31.65%	31.60%	31.63%	**37.56%**	33.44%
		76.82%	76.65%	76.62%	76.60%	76.63%	67.85%	74.58%

is *vital* when it comes to measuring human understanding [9]. For the quantitative analysis, we utilize *Edit Distance (ED)* and *Jaccard Similarity Coefficient (JSC)* metrics for evaluating the similarity between the original/legitimate texts and the texts given as a response by the human respondents. In this way, we are able to conclude about whether or not, the human readers are able to interpret (and with what ease) the given adversarial texts.

After gathering the responses from *51* participants we came up with the following graphs: (a) average difficulty of interpreting all given perturbed texts (Fig. 6(a) red bars), (b) most difficult to be interpreted question (Fig. 6(a) blue bars), (c) least difficult to be interpreted question (Fig. 6(a) green bars), (d) interpretation difficulty distribution (Fig. 6(b)), and (e) average ED and JSC for each question (Fig. 7).

As shown in Figure 6(a) (red bars), the largest percentage of the respondents, i.e., *91.50%*, answered that the questions were "very easy" or "easy" to interpret. In fact, only the *2.39%* of the respondents found the questions "difficult" or "very difficult" to interpret.

In Figure 6(a) (blue bars), the *most difficult* to be interpreted question, along with its interpretation difficulty, is shown. Those results were expected as TPT 1 indeed perturbs the most, a given legitimate text. Nonetheless, even with this TPT the largest percentage of the respondents, i.e., *62.74%*, found the perturbed texts "very easy" or "easy" to interpret, whereas only the *11.76%* of the respondents found the perturbed texts "difficult" or "very difficult" to interpret. It is worth noting that even the respondents that found specific adversarial texts "difficult" or "very difficult" to recognize have *eventually interpreted them correctly*. In other words, *100%* of the respondents successfully interpreted all adversarial texts produced by our ATGF.

In Figure 6(a) (green bars), the *least difficult* to be interpreted question, along with its interpretation difficulty, is shown. Again, those results were expected as inserting spaces between negative/positive sentiment or toxic/non-toxic words' characters, i.e., TPT 6, indeed perturbs the *least* a given legitimate text. This TPT demonstrates excellent human readability preservation as only *2%* of the respondents answered that the perturbed text "The sort of "little" film which studios used to e x c e l at but seldom make anymore" was "very difficult" or "difficult" to interpret.

Figure 6(b) shows the *interpretation difficulty distribution* of the responses gathered. As shown, the highest concentration level

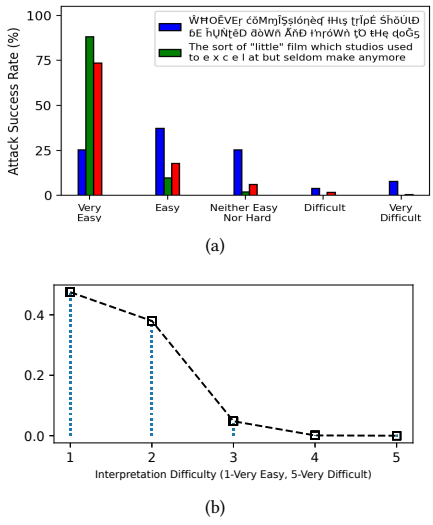
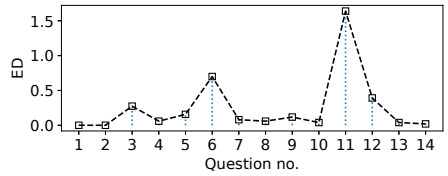


Figure 6: Average interpretation difficulty for the most difficult (6(a)-blue bars), least difficult (6(a)-green bars), and all given perturbed texts (6(a)-red bars), and the distribution of interpretation difficulty faced when recognizing the given adversarial texts 6(b).

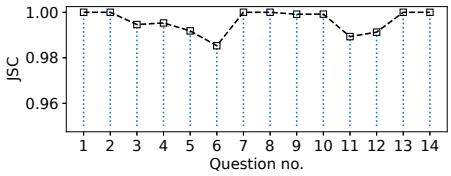
peaks at the "very easy" response and decreases as it moves to the more difficult to interpret answers. The majority of the respondents considers the perturbed texts "very easy" or "easy" to interpret, thus, demonstrating EvilText's capabilities on preserving the original text's semantics and human readability.

Figure 7(a) shows that the *average ED* for each question is very low, meaning that *minor* modifications, i.e., insertions, deletions, or substitutions, have to be made in order for the actual/original non-perturbed text to match the given by the respondents text. The highest average ED is *1.64* for the most difficult question shown in Figure 6(a). However, even the highest average ED, i.e., 1.64, is very low, meaning *very high similarity* between the original and the given by the respondent, texts. Figure 7(b) shows that the *average JSC* for each question is *very high*, the lowest being *98.53%*. Again, this means that *minor* differences, between the original/non-perturbed and the given by the respondent, texts, exist. However, even *98.53%* JSC is considered to be a *very high* similarity percentage.

[9]Note that we are not obliged to get an IRB approval as we do not collect any demographic information which might be sensitive.



(a) Average ED



(b) Average JSC

Figure 7: Average ED and JSC for each question (14 in total).

EvilText *outperforms* state-of-the-art ATGFs in terms of original text's semantics and human readability preservation. In particular, *100%* of the presented adversarial text has been *correctly* interpreted by the human participants. In addition, the *average ED* and *JSC* for all questions is *0.21* and *99.67%*, respectively, which is a clear evidence regarding the original text's semantics and human readability preservation capabilities of EvilText ATGF. Observing all the results reported in this section, we conclude that EvilText is an effective and efficient ATGF that can successfully evade some of the most popular NLP machines having *little impact* on either the human understanding or the original text's semantics.

Transferability. The *transferability* property of adversarial samples is of major importance for effective ATGFs [46]. The adversarial samples crafted for a specific NLP machine M_1 will also be misclassified by any other NLP machine M_2 ($M_1 \neq M_2$), as long as the transferability property holds between M_1 and M_2 [36]. In other words, if the transferability property holds, the attacks offered from a particular ATGF can be also extended to any other potential system. Thus, we advise NLP services' operators to take our ATGF's attacks into explicit consideration.

The attack success rates reported in Tables 4, 5 & 7, showcase the transferability of our attacks on a variety of target white-box/black-box NLP machines. our attacks are *black-box* based, thus, *not* tailored to any specific NLP machine or dataset. In addition, EvilText does not require the confidence score neither the output label from the target model, in constrast to other approaches found in the literature [16, 30, 31]. As a result, EvilText can target a wide-range of NLP systems without the need of any tuning or modification.

5 POTENTIAL DEFENCES

We suggest 3 defence strategies that could be followed in order to mitigate, but not eliminate, the risk against our attacks. The effectiveness, however, of each defence mechanism (and potential combinations of them) is out of scope; we leave this as a future research direction.

Spelling Check. Context-aware spelling checkers, such as Microsoft Azure[10], can significantly enhance the performance of NLP systems detecting, in most cases, adversarial text crafting methods based on spelling errors. In its simplest form, a spell checker will essentially restrict the use of sentences that contain spelling errors. Nonetheless, this is not a good practise as human writers are very prone to (unintentional) spelling errors. Thus, employing such a technique will result in a high percentage of *false positives*. Moreover, spell checking *cannot* be used to prevent sophisticated attacking schemes for it is trivial to create ambiguous sentences that can result in *false positives* by changing even single words in a document. For instance, a possible spelling correction may result in a wrong sentiment prediction by (falsely) correcting a wrongly spelled word.

Li et al. [30] showed that spelling checkers cannot mitigate the risk against sophisticated TPTs which are not based on misspelling perturbations. Testing the aforementioned spell checker we observe the same results as Li et al. In particular, Microsoft Azure spell checker affects the performance of TPTs that do not make use of the unicode map, i.e., TPTs 4 and 6, due to their wrong spelling nature. However, the rest of the TPTs remain *unaffected*.

Adversarial Training. In adversarial training we train a particular classifier with, in addition to legitimate-ones, adversarial input samples. As a result, the DL-based classifier will exhibit increased performance on classifying adversarial texts [30]. However, the three limitations of this defence method are: (a) the attackers do not make their approaches of crafting adversarial text public and thus adversarial training does not perform well on unknown adversarial attacks, (b) a large number of adversarial text samples, created using as many as possible TPTs, is needed for the training process and most importantly (c) adversarial training might reduce the model's performance, i.e., its accuracy when no adversary exists [41]. In addition, the literature showed that it is rather difficult to produce a robust network using adversarial training, even for black-box settings where the attacker has restricted information on the target model [39]. Thus, the aforementioned facts along with the substantially large space of possible TPTs make the use of this defence strategy a challenging process.

However, we plan to explore the potential effectiveness of adversarial training when the defender has access to data samples which are crafted using: (a) different from our TPTs, (b) some different and some similar to our TPTs, and (c) the same as our TPTs. In addition, the volume of adversarial training instances required to effectively detect such adversarial samples should be estimated. We leave this as a future research direction.

Replacing Unknown Words. Our unicode map (see Section 2) can be utilized (in reverse order) to replace the perturbed characters with the original English alphabet letters in order to craft the initial non-perturbed text. As a result, the NLP models will (most probably) correctly detect the sentiment/toxicity of a given input text considering all words' embeddings. Thus, this defence strategy, which is essentially the equivalent of *restricting* the use of unicode characters, is a potentially effective, yet intrusive, mechanism for mitigating our attacks. Finally, this defence strategy highlights the

[10] https://azure.microsoft.com/zh-cn/services/cognitive-services/spell-check/.

contribution of delivering such a similarity map for mitigating such security risks.

6 DISCUSSION

Sum up. Observing the results reported in Tables 4, 5 and 7 one can easily see that the attack success rates vary across different TPTs and target models. This is because different TPTs affect different target ML models (due their underlying structure). Thus, we deliberately design EvilText's architecture in such a way for allowing attackers to easily extend the offered TPTs with additional TPTs (or combine existing TPTs) that evade specific target NLP systems.

The average attack success rate on negative sentiment reviews contained in the three benchmark datasets, i.e., MR_v1.0, MR_v2.0, and IMDB, is *10.75% higher* than the attack success rate on positive sentiment reviews. This indicates that *negative sentiment* reviews are *more prone* to perturbation attacks than positive sentiment reviews. The same fact holds for the toxic content detection dataset as the average attack success rate on toxic samples is *41.46% higher* than the attack success rate on non-toxic samples. Interestingly, our attacks are more effective on *negative sentiment/toxic* samples rather than positive sentiment/non-toxic samples.

Limitations. For EvilText to be successfully applied, the positive/negative opinion words lists must be available. Nonetheless, this might not be the case in other application scenarios, where such information is not available to the adversary. Furthermore, EvilText, similar to TextBugger, DeepFool and DeepWordBug, is limited in the two-class sentiment/toxicity setting; a more sophisticated attack strategy is required in the multiclass setting. This is because in the multiclass setting, it can be more difficult to identify which words need to be changed and how they need to be changed for achieving the desired misclassification.

7 FUTURE DIRECTIONS

Other, similar to ours, TPTs that may improve EvilText's performance should be defined and evaluated. For example, TPTs that only perturb a fraction of the sentiment/toxicity defining words' letters should be evaluated for their performance, in terms of both evasion success rate and human readability preservation. Moreover, sophisticated attacking methods that will infiltrate to an existing black-box NLP machine, exposing valuable to the attacker information about the model's internal structure and hyper-parameter values, should be explored. Furthermore, our TPTs should be evaluated against recent transformer-based target NLP models, such as BERT [10], which demonstrate high performance in similar NLP tasks. In addition, an empirical evaluation for the performance of the suggested defences (and potential combinations of them), in terms of mitigating such TPTs, is considered critical to be performed. Nonetheless, in order to develop a generic defence strategy that effectively protects NLP systems from a wide-range of similar adversarial attacks the entire (or a large portion) of the possible attacks spectrum has to be explored. Finally, our attacks are oriented only in the English language and thus future extensions of EvilText should also include attacks on other languages.

8 RELATED WORK

Gao et al. [16] and Liang et al. [31] focus on the development of ATGFs which are mainly based on inserting extra characters, modifying and/or deleting existing ones. Li et al. [30] proposed an ATGF, namely TextBugger, which perturbs input samples by deliberately misspelling important words, and by replacing a few words, which are obtained by nearest neighbour searching in the embedding space, without changing the original meaning. Our ATGF not only *outperforms* TextBugger in effectiveness but also in *efficiency*. In particular, EvilText is *substantially more efficient* than TextBugger as we eliminate the need for iteratively querying the target NLP model (each time excluding one word) for detecting the sentiment/toxicity defining words. [16, 30] and [31] evaluate their attacks in both white-box and black-box attack settings, whereas [14, 38] and [43] evaluate their attacks in white-box setting only. In addition, [16, 30, 31] focus *solely* on the negative sentiment/toxic sample to positive sentiment/non-toxic sample type of attack, whereas in our case we also examine the opposite one. Some works generate adversarial texts by replacing a word with one legible but OOV word [4, 16, 20], whereas others utilize genetic algorithms for replacing words only with semantically similar ones [1]. Cheng et al. [8] proposed adversarial attacks targeting, however, sequence-to-sequence NLP models. Finally, some methods enrich the legitimate text with the addition of distracting sequences of text for causing a misclassification [23]. However, these methods have significantly high overhead as intensive manual effort is required for smoothening the perturbed text meaning.

9 CONCLUSION

EvilText is an effective and efficient ATGF used to perturb legitimate text for causing some of the most popular NLP machines to misbehave. It generates high-quality adversarial texts that successfully evade NLP systems, being at the same time painless for humans to interpret. This fact, in combination with the high transferability of our attacks, disclose great risks for many real-world NLP text-filtering systems.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for helping us to improve the final version of this paper. In addition, we thank Stelios Tymvios for his contribution to early drafts of this paper. This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreements No. 830929 (CyberSec4Europe) and the Marie Skłodowska-Curie grant agreement No. 101007673.

REFERENCES

[1] 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* (2018).
[2] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2017. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397* (2017).
[3] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. 2006. Can machine learning be secure?. In *ASIACCS*. ACM, 16–25.
[4] Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173* (2017).
[5] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2011. Design of robust classifiers for adversarial environments. In *SMC*. IEEE, 977–982.

[6] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[7] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *EuroS&P*. IEEE, 39–57.

[8] Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. 2018. Seq2Sick: Evaluating the Robustness of Sequence-to-Sequence Models with Adversarial Examples. (2018). arXiv:cs.LG/1803.01128

[9] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. 2017. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373* (2017).

[10] 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* (2018).

[11] Antreas Dionysiou, Michalis Agathocleous, Chris Christodoulou, and Vasilis Promponas. 2018. Convolutional Neural Networks in Combination with Support Vector Machines for Complex Sequential Data Classification. In *ICANN*. Springer, 444–455.

[12] Antreas Dionysiou and Elias Athanasopoulos. 2020. SoK: Machine vs. Machine–A Systematic Classification of Automated Machine Learning-Based CAPTCHA Solvers. *Computers & Security* (2020), 101947.

[13] Antreas Dionysiou, Vassilis Vassiliades, and Elias Athanasopoulos. 2021. *HoneyGen: Generating Honeywords Using Representation Learning*. Association for Computing Machinery, New York, NY, USA, 265–279. https://doi.org/10.1145/3433210.3453092

[14] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (2018), 31–36.

[15] Anthony Y Fu, Xiaotie Deng, Liu Wenyin, and Greg Little. 2006. The methodology and an application to fight against unicode attacks. In *SOUPS*. 91–101.

[16] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *SPW*. IEEE, 50–56.

[17] Zhitao Gong, Wenlu Wang, Bo Li, Dawn Song, and Wei-Shinn Ku. 2018. Adversarial texts with gradient methods. *arXiv preprint arXiv:1801.07175* (2018).

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

[19] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH*. ACM, 18.

[20] Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138* (2017).

[21] Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *SIGKDD*. ACM, 168–177.

[22] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. 2011. Adversarial machine learning. In *AISec*. ACM, 43–58.

[23] Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *in EMNLP* (2017), 2021–2031.

[24] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[25] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *in EMNLP* (10 2014), 1746–1751.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*. 1097–1105.

[27] Brandon Laughlin, Christopher Collins, Karthik Sankaranarayanan, and Khalil El-Khatib. 2019. A Visual Analytics Framework for Adversarial Text Generation. *arXiv preprint arXiv:1909.11202* (2019).

[28] Yann LeCun, Yoshua Bengio, et al. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 3361, 10 (1995), 1995.

[29] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. 1995. Comparison of learning algorithms for handwritten digit recognition. In *ICANN*, Vol. 60. Perth, Australia, 53–60.

[30] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. *in NDSS* (2019). https://doi.org/10.14722/ndss.2019.23138

[31] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep Text Classification Can Be Fooled. *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (2018), 4208–4215.

[32] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *49th ACL: Human language technologies*. ACL, 142–150.

[33] Bo Pang and Lillian Lee. 2004. A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. In *42nd ACL*. ACL, 271–es.

[34] Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *43rd ACL*. ACL, 115–124.

[35] Bo Pang and Lillian Lee. 2008. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval* 2 (2008), 1–135.

[36] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *ASIACCS*. ACM, New York, NY, USA, 506–519.

[37] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *EuroS&P*. IEEE, 372–387.

[38] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting adversarial input sequences for recurrent neural networks. In *MILCOM*. IEEE, 49–54.

[39] Sanglee Park and Jungmin So. 2020. On the Effectiveness of Adversarial Training in Defending against Adversarial Example Attacks for Image Classification. *Applied Sciences* 10, 22 (2020), 8079.

[40] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. 1532–1543.

[41] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. 2019. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032* (2019).

[42] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[43] Suranjana Samanta and Sameep Mehta. 2017. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812* (2017).

[44] David Sculley, Gabriel Wachman, and Carla E Brodley. 2006. Spam Filtering Using Inexact String Matching in Explicit Feature Space with On-Line Linear Classifiers.. In *TREC*.

[45] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[46] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil Jain. 2019. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv preprint arXiv:1909.08072* (2019).

[47] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*. 649–657.

[48] Ye Zhang and Byron Wallace. 2017. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In *IJCNLP*. AFNLP, Taipei, Taiwan, 253–263.

[49] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating natural adversarial examples. *in ICLR* (2018).