The Pennsylvania State University
The Graduate School

TRUSTWORTHY MACHINE LEARNING: LEARNING UNDER SECURITY, EXPLAINABILITY AND UNCERTAINTY CONSTRAINTS

A Dissertation in
Information Sciences and Technology
by
Thai Le

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2022

The dissertation of Thai Le was reviewed and approved* by the following:

Dongwon Lee
Professor of College of Information Sciences and Technology
Dissertation Adviser, Chair of Committee

Suhang Wang
Assistant Professor of College of Information Sciences and Technology

Prasenjit Mitra
Professor of College of Information Sciences and Technology

S. Shyam Sundar
James P. Jimirro Professor of Media Effects
Professor of Donald P. Bellisario College of Communications

Mary Beth Rosson
Professor of College of Information Sciences and Technology
Graduate Program Chair

Abstract

Trustworthy machine learning models are ones that not only have high accuracy but also well perform under various realistic constraints, security threats, and are transparent to users. By satisfying these constraints, machine learning models can gain trust from their users and thus make it easier for them to be adopted in practice. This thesis makes contributions on three aspects of trustworthy machine learning, namely (i) *learning under uncertainty*–i.e., able to learn with limited and/or noisy data, (ii) *transparent to the end-users*–i.e., being explainable to the end-users, and (iii) *secured and resilient machine learning*–i.e., adversarial attacks and defense from/against malicious actors. Particularly, this thesis proposes to overcome the lack of high-quality labeled textual data that is necessary for training effective ML classification models by directly synthesizing them in the data space using generative neural networks. Moreover, this thesis designs a novel algorithm that facilitates accurate and effective post-hoc explanations of neural networks' predictions to the end-users. Furthermore, this thesis also demonstrates the vulnerability of a wide range of fake news detection models in the literature against a carefully designed adversarial attack mechanism where the attackers can promote fake news or demote real news on social media via social discourse. This thesis also proposes a novel approach that adapts the "honeypot" concept from cybersecurity to proactively defend against a strong universal trigger attack. Last but not least, this thesis contributes to the adversarial text literature by proposing to study, extract and utilize not machine-generated but realistic human-written perturbations online. Through these technical contributions, this thesis hopes to advance the adoption of ML systems in high-stakes fields where mutual trust between humans and machines is paramount.

Table of Contents

List of Figures

List of Tables

Acknowledgments

This material is based upon work supported by the NSF under Award No. #1742702, #1820609, #1909702, #1915801, #1934782, #1940076, #2114824. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the agencies.

Dedication

I dedicate my dissertation work to my family and friends, especially my beautiful wife–Ashlinn, my parents–Thu and Dung, my sister–Duong, and my mentors–a.k.a “senior" friends–Ken, Joyce, Bob, and Marleen. I also want to dedicate this dissertation to one of my high-school teachers–Mrs. Khanh Van, and my college thesis advisor–Prof. Pishva Davar, who cultivated the very early research interests in me. Finally and most importantly, I dedicate my dissertation to my God Almighty, from whom I got all the strengths, perseverance, wisdom, and creativity.

Chapter 1
Introduction

1.1 Trustworthy Machine Learning

Throughout the years, machine learning (ML) models have continuously broken their records in terms of prediction performance in a wide range of fields such as visual objects recognition and natural language understanding. Their prediction performances are usually evaluated by how accurately they can perform on unseen data–i.e., often called test sets. Until recently, prediction accuracy on test data has been a gold standard to measure the performance of ML models. However, as ML models are prevalently integrated into every aspect of our daily lives, which can range from cancer diagnosis to potential self-driving automobiles, answers to the question: *whether ML models are secure and reliable?* are much needed. Therefore, this thesis proposes to study the trustworthiness of ML models on three aspects (Figure 1.1), namely (i) *learning under uncertainty*–i.e., able to learn with limited and/or noisy data, (ii) *explainability for end-users*–i.e., being explainable and transparent to end-users, and (iii) *adversarial ML*–i.e., adversarial attacks and defense from/against malicious actors. These aspects represent several challenges when examining the relationship between a ML model and its data source (Figure 1.1A), its end-users (Figure 1.1B) and its adversaries (Figure 1.1C), respectively. Through technical contributions in these areas, this thesis encourages the adoption of ML systems in high-stakes fields where mutual trust between humans and machines is paramount. Also, it provides the regulators with frameworks to assess the security and transparency of novel ML systems. This chapter will summarize each of the three trustworthy aspects of ML and their respective proposed technical

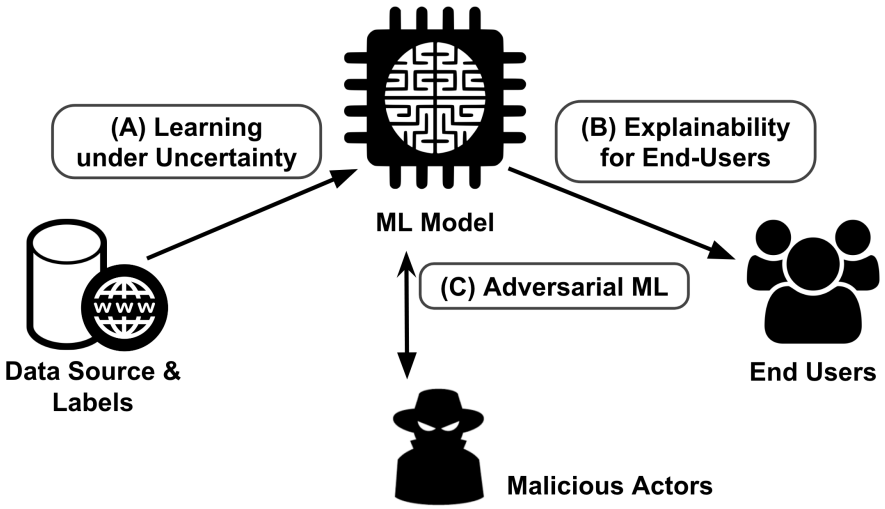


Figure 1.1: Three aspects of trustworthy ML.

contributions of the thesis.

1.1.1 Learning under Uncertainty

Supervised ML models require learning from labeled datasets. Particularly, in classification tasks, this dataset either includes positive and negative labels–i.e., in binary classification such as fake-news prediction, or positive labels of different categories–i.e., in multinomial classification such as news categorization. In many critical domains, labeled data for training classification ML models that are curated mainly through human annotators can be limited and/or noisy. This is often due to either the nature of a specific domain–e.g., there is less fake than real news, and/or the discrepancy in subjective judgment among the annotators–e.g., clickbait v.s. non-clickbait headlines. To overcome these issues, especially when there is a lack of high-quality labels, one can easily over-sample an existing labeled dataset using statistical methods such as SMOTE [1] on the feature space–e.g., Euclidean space. In the natural language processing (NLP) domain, however, this approach is limited due to possible loss of information when encoding texts from the data space–i.e., texts, to the feature space–i.e., vectors. This also means that over-sampled examples on the vector space do not necessarily correspond to meaningful texts. Moreover, this does not take into account the noisiness of the input labeled

Icons of Fig. 1.1 are retrieved from pngegg.com, icon-library.com, pikpng.com, iconspng.com.

dataset. Thus, chapter 2 proposes to utilize generative models to over-sample labeled data directly on the data space or to synthesize texts [2]. My experiments on clickbait texts show that Variational AutoEncoder (VAE)-based generative models can capture the most consistent characteristics of the input labeled texts–i.e., denoising, and generate synthetic clickbaits that consistently improves the accuracy of various ML clickbait detectors by up to 15%, outperforming even the best solution in the *Clickbait Challenge* in 2017 . Especially, the generated clickbaits are also highly overlapped with the original texts in the NLP feature space.

1.1.2 Explainable AI (XAI) for End-Users

Beyond delivering accurate predictions on unseen examples, ML models also need to communicate to the end-users why they made such predictions. This applies especially to large neural network models where their models' weights are often too complex to interpret. Although there have been several works in enabling the interpretability of neural network models, their target audiences are often skewed towards researchers and developers, who already have some experience in ML/AI. Chapter 3 fills in this gap by developing a novel algorithm to explain AI systems specifically for the end-users, who often do not have a relevant technical background. Explaining predictions of an ML model to the end-users requires two main components, namely the mode of the explanation–e.g., predictive strengths of features, and how to present it. This becomes especially challenging when the domain datasets are tabular and in high-dimensional vectorized formats. This is because there can be hundreds or thousands of features to select to present to the users, and presentation means such as highlighting a patch of an image or a span of a sentence is also limited in the tabular domain. To resolve this, this thesis borrowed two notable ideas, namely "explanation by intervention" from causality and "explanation is contrastive" from philosophy, and proposed a novel solution, named as GRACE [3], that better explains NN models' predictions for tabular datasets. In particular, given a model's prediction as label X, GRACE *intervenes and generates a minimally-modified contrastive sample to be classified as Y*, with an intuitive textual explanation, answering the question of "Why X rather than Y?" Human studies show that GRACE can generate explanation texts that are not

`https://webis.de/events/clickbait-challenge/`

only more friendly and understandable but also help the users make more accurate decisions than the strong baseline LIME [4].

1.1.3 Adversarial Attack and Defense

Similar to other technologies, ML models are not free from malicious attacks when they are deployed in practice. This thesis investigates such an attack in the case of ML fake-news detection models. Text commentaries play a vital role in online discourse. They are also strongly predictive features of ML models in high-stake domains such as fake news detection. Thus, this thesis hypothesized that the adversaries can leave malicious comments on social posts, *without* the need to change their original contents, to attack these ML models. As an illustration, chapter 4 introduces MALCOM [5], a novel conditional text generation algorithm that can generate high-quality and coherent adversarial comments for a given post. Once attached to a social post, these comments can force many previously published fake news detectors to predict it as fake or real news with over 90% accuracy on average. This thesis also demonstrates the possibility of robustly defending against such malicious attacks. Especially, related works often characterize and detect adversarial attacks *only after* they happen. As in the saying "an ounce of prevention is worth a pound of cure", this thesis investigates to adopt the "honeypots" concept from *cybersecurity* to trap potential attacks–i.e., to *proactively* defend against these attacks even before they happen. To do this, chapter 5 introduces DARCY [6], a novel algorithm that greedily searches and injects multiple trapdoors into an NN model to "bait and catch" potential attacks with an accuracy of over 99% on average. Intuitively, these trapdoors are artificial local minimum on the target model's loss landscape, which makes it very attractive for the adversaries to fall in when they try to optimize the attacks. Furthermore, previous proposals on adversarial attacks often hypothesize different adversarial attack strategies based on some known vulnerabilities of textual ML models. However, text perturbations in real-life texts are much more diverse and nuanced–e.g., "democrats"→"demokRATs". To derive a more realistic attack, chapter 6 proposes an *inductive approach* to mine text perturbations in the wild [7]. The extracted human-written perturbations is then utilized to derive a more realistic attack and to enhance the robustness of a BERT classifier against noisy human-written texts online.

Chapter 2
Learning under Uncertainty: Synthesizing High-Quality Labeled Texts to Improve Clickbait Detection

2.1 Background

In Feb. 2018, US President Donald Trump posted on Twitter: *"NEW FBI TEXTS ARE BOMBSHELLS!"*, which has drawn much attention from the public and media. This tweet exhibits many characteristics of *clickbaits*–i.e., catchy social posts or sensational headlines that attempt to lure readers to click. Other examples of clickbaits can be found in Table 2.1.

Clickbaits often hide critical information or fabricate the contents on the landing pages by using exaggerated or catchy wording. Yet, social media has made it possible for clickbaits to quickly go viral, thus a potential means of spreading misinformation.

Table 2.1: Human-written & machine-generated clickbaits

Human-written clickbaits
Pregnant mother of 12 accused of keeping kids in waste-filled...
54 facts that will change the way you watch disney movies
Who, What, Why: How do dogs donate blood?
Machine-generated clickbaits (by $\mathcal{G}_{\text{VAE}}$)
5 ways criminals will try to scam you about tax this summer
We know your personality based on which cat you choose
Why you should be proud to be naked
Machine-generated clickbaits (by $\mathcal{G}_{\text{infoVAE}}$)
43 magical moments that made game and thanksgiving
29 things every student has while in college
29 ridiculously posts about the damn disasters

Indeed, a Facebook media analysis [8] shows that a clickbait post receives more attention via shares and comments than a non-clickbait one.

Viewed as one type of *fake news* in a broad sense [9],

clickbaits not only frustrate readers, but also violate the journalistic code of ethics [10]. Scholars have argued that the current trend toward merging commercial and editorial interests by means of clickbaits is severely detrimental to the overall information ecosystem, particularly posing a threat to societal/democratic values [11].

Therefore, it has become critically important to develop proactive solutions to the use of clickbaits. By and large, existing approaches have tended to focus on the "*postmortem*" approach–i.e., assuming that clickbaits are out there, how to develop computational solutions to best detect them (e.g., [8, 12–14]). While these prior works are important and effective, their performance is highly dependent on the quantity and quality of training datasets available. However, there is a paucity of high-quality labeled training datasets that are heterogeneous in sources and large in quantity. This is because annotating labels is expensive, and most of existing datasets are passively collected from external sources, instead of actively generated. To overcome this problem at large, we propose a research question: *how to generate new headlines and titles that resemble real-life clickbaits and how to use them as additional training samples to improve clickbait detection models?*

To systematically study this question, we commission various human entities (e.g., crowdworkers and journalism students) and deep generative models in simu-

lated experiments to generate clickbaits from scratch. We refer to these clickbaits that are generated under *simulated intent, i.e., to generate attractive headlines for news articles*, as **synthetic clickbaits**. Furthermore, by simulating a similar intent in generating attractive headlines, we hypothesize that synthetic clickbaits might share some similarities in the use of language (e.g., writing style, word choice, grammar patterns) with clickbaits collected in real life, which might be different from that of non-clickbait. We tested this proposition by examining whether generated synthetic clickbaits can be used as additional training examples to strengthen classification performance.

We also want to compare these synthetic clickbaits with synthetic data sampled by Synthetic Minority Over-sampling Technique (SMOTE) in terms of predictability, how they capture the original NLP features in terms of distribution, robustness, and interpretability.

Considering that generated clickbaits can resemble various characteristics of real clickbaits, malicious publishers might take advantage of different entities (e.g., machines) to generate vast amount of clickbaits to disseminate low-quality content just to attract online traffic. As a proactive defense against this potential, we further examine the use of Machine Learning (ML) models to verify news sources, in order to differentiate various types of clickbaits. Formally, we propose the following research questions:

RQ1 How do we generate synthetic clickbaits from raw training samples as additional training samples to improve supervised-learning clickbait detection models?

RQ2 What are the differences among synthetic clickbaits generated by humans, generative models, and statistical method in terms of predictive power, NLP feature encapsulation, robustness and interpretability?

RQ3 How can we differentiate clickbaits based on sources (e.g., machine versus human-written clickbaits)?

By answering these research questions, this chapter makes the following contributions:

- Overcoming the lack of labeled training samples by exploiting human and deep generative models, we generate diverse types of synthetic clickbaits.

Table 2.2: Statistics of five types of synthetic clickbaits

Statistics	$\mathcal{P}$	$\mathcal{M}$	$\mathcal{C}$	$\mathcal{S}$	$\mathcal{A}$
Avg # words	10.27	11.00	11.83	8.61	10.18
Std. Dev.	4.34	4.39	4.44	2.79	2.94
Avg # chars	46.81	51.95	56.6	42.32	44.69
Std. Dev.	12.61	19.96	21.12	20.32	12.48

- We demonstrate that using both raw and synthetic clickbait samples (generated from raw samples) consistently improve clickbait detection models by up to 14.5% in AUC, even outperforming SMOTE and two top-performed clickbait detection algorithms from Clickbait Challenge 2017.
- Leveraging deep learning models, we generate more interpretable oversampling data that also better capture the distribution of NLP-based domain knowledge from original clickbaits compared with SMOTE.
- We demonstrate that the clickbaits generated by different entities have significant differences in features so that ML models can differentiate them with an accuracy of 20%–39% higher than random guesses.

2.2 RQ1: Generating Synthetic Clickbaits

We begin with two raw datasets, $\mathcal{P}$ and $\mathcal{M}$, representing two dominant sources of clickbaits prevalent today–i.e., mainstream news media and general social media, respectively [8]. Then, to generate synthetic clickbaits from raw datasets, we explore two types of human sources (i.e., crowdworkers as novice users and journalism majors as domain experts) and VAE-based generative models.

Table 2.2 compares the lengths of synthetic clickbaits from all five entities.

2.2.1 Crowdworkers-Generated Clickbaits $\mathcal{C}$:

To collect clickbaits generated by crowdworkers, we utilize the Amazon MTurk (AMT) platform. From the articles used in the Clickbait Challenge 2017, we first filtered out very short articles with less than 50 words in content. In addition, for very long articles with more than 500 words in content, we presented only

Table 2.3: Experiment datasets

Dataset	Description	#Pos	#Neg
$\mathcal{P}_{train}$	Training set from $\mathcal{P}$	2,239	11,201
$\mathcal{P}_{test}$	Testing set from $\mathcal{P}$	960	4,800
$\mathcal{M}_{train}$	Training set from $\mathcal{M}$	3,681	11,337
$\mathcal{M}_{test}$	Testing set from $\mathcal{M}$	1,578	4,859
$\mathcal{C}$	Training set from workers	778	0
$\mathcal{S}$	Training set from Students	785	0
$\mathcal{A}^{P}_{\text{VAE}}$	$\mathcal{G}_{\text{VAE}}$ trained on $\mathcal{P}_{train}$	8,962	0
$\mathcal{A}^{M}_{\text{VAE}}$	$\mathcal{G}_{\text{VAE}}$ trained on $\mathcal{M}_{train}$	7,656	0
$\mathcal{A}^{\mathcal{P}}_{\text{infoVAE}}$	$\mathcal{G}_{\text{infoVAE}}$ trained on $\mathcal{P}_{train}$	8,962	0
$\mathcal{A}^{\mathcal{M}}_{\text{infoVAE}}$	$\mathcal{G}_{\text{infoVAE}}$ trained on $\mathcal{M}_{train}$	7,656	0
$\mathcal{O}^{\mathcal{P}}$	SMOTE on $\mathcal{P}_{train}$	8,962	0
$\mathcal{O}^{\mathcal{M}}$	SMOTE on $\mathcal{M}_{train}$	7,656	0

the first 500 words to reduce the amount of reading for workers. As the first 3-4 paragraphs of news articles often summarize the content, the first 500 words sufficiently captured the gist of the articles. Then, we recruited AMT workers located in US (who are more likely to be familiar with the topics of the articles) with approval rates > 0.95.

In the MTurk task, next, we first showed a Wikipedia link with the definition of clickbait,

but did not provide additional information that might influence the way workers generated clickbaits. Second, for each article shown, we asked workers to read the article and write a clickbait headline, with no more than 25 words. In the end, 85 workers generated 778 clickbait headlines for 200 selected articles. This provided us a total of 62 articles with 3 different clickbait headlines, 113 articles with 4 clickbaits, 10 articles with 5 clickbaits, and 15 articles having 6 clickbaits.

2.2.2 Student-Generated Clickbaits $\mathcal{S}$:

Another source for headline creation was undergraduate students who are being trained to learn about the art and craft of journalistic writing and reporting. We recruited participants from 8 different classes at a large northeastern university in US. Participants received extra course credit for their participation. Because

https://en.wikipedia.org/wiki/Clickbait

we wanted to include participants with different levels of expertise, we recruited from 3 lower-level classes and 5 upper-level classes. Participants in the lower-level classes represent *amateurs* who are beginning to learn about the journalistic style of writing, whereas those from the upper-level classes represent students who are *semi-experts* and have an advanced understanding of the principles of reporting and headline creation. A total of 125 students participated (i.e., 76.8% and 23.2% from lower- and upper-level classes, respectively).

The design principle and articles used to generate these headlines were the same as the ones used for AMT participants. We first provided students with a definition of clickbait, without providing additional information, and asked them to generate a clickbait headline, with no more than 25 words.

Each student completed an average of 6 headlines, ranging from 1 to 22. The students generated 785 clickbaits in total.

2.2.3 Algorithm-Generated Clickbaits $\mathcal{A}$:

Due to recent advancements in generative models, next, we turn to machine-generated clickbaits. We utilize different variations of VAE-based generative models in the task of generating synthetic clickbaits. VAE-based generative models are selected because the latent code z learned from the model appears to encapsulate information about the number of tokens, and their parts of speech (POS) and topics [15], all of which are shown to be effective predictive NLP features in differentiating clickbaits from non-clickbaits (e.g., [10, 14, 16]).

We utilized the two generative models, namely VAE and infoVAE, as introduced in [15,17] to generate synthetic clickbaits. While the first model uses original VAE loss function [15], the second model uses Maximum Mean Discrepancy with a Gaussian Kernel [17] to replace the original KL divergence term. We denote the synthetic datasets generated by the two models $\mathcal{A}_{\text{VAE}}$ and $\mathcal{A}_{\text{infoVAE}}$ respectively. We refer the readers to their original papers for objective functions formulation and as well as optimization techniques. Table 2.1 lists some the examples of clickbaits generated by the two models trained on a subset of clickbaits drawn from $\mathcal{M}$ and $\mathcal{P}$.

2.3 RQ2: Assessing Synthetic Clickbaits

In this section, we seek to differentiate *synthetic clickbaits* generated by humans, generative models, and SMOTE with four analytic questions (AQs) as follows:

AQ1 **Predictive Power**: How much do synthetic clickbaits help improve ML models in detecting clickbaits?

AQ2 **NLP Encapsulation**: How well do synthetic clickbaits encapsulate NLP feature distribution from the training dataset?

AQ3 **Robustness**: What is the minimum amount of synthetic clickbaits needed to improve ML clickbait detection model? Is such an improvement proportional to the increase in synthetic clickbaits?

AQ4 **Interpretability**: Are synthetic clickbaits interpretable to humans?

Table 2.4: Mean AUC scores and their relative changes (%) on $\mathcal{P}_{test}$ using different *oversampling* methods.

	Baseline	Synthesized Supervised Learning (SSL)				SMOTE
Algorithms	$\mathcal{P}_{train}$	$\mathcal{P}_{train} \cup \mathcal{C}$	$\mathcal{P}_{train} \cup \mathcal{S}$	$\mathcal{P}_{train} \cup \mathcal{A}^{\mathcal{P}}_{\text{VAE}}$	$\mathcal{P}_{train} \cup \mathcal{A}^{\mathcal{P}}_{\text{infoVAE}}$	$\mathcal{P}_{train} \cup \mathcal{O}^{\mathcal{P}}$
AdaBoost	0.88	0.88 (-0.11%)	0.88 (-0.11%)	**0.91 (+2.79%)**	0.90 (+1.98%)	0.89 (+1.23%)
Bagging Clf	0.88	0.89 (+0.74%)	0.89 (+0.67%)	**0.91 (+3.05%)**	0.90 (+1.98%)	0.88 (+0.22%)
Decision Tree	0.86	0.87 (+0.71%)	0.87 (+0.66%)	**0.89 (+2.95%)**	0.87 (+0.62%)	0.86 (+0.33%)
GradientBoosting	0.89	0.89 (-0.32%)	0.89 (-0.16%)	**0.92 (+3.13%)**	0.91 (+1.94%)	0.90 (+1.23%)
KNeighbors Clf	0.83	0.83 (+0.08%)	0.83 (+0.08%)	**0.88 (+6.89%)**	0.86 (+4.64%)	0.87 (+5.06%)
Logistic Regression	0.91	0.90 (-0.69%)	0.90 (-0.69%)	**0.92 (+1.48%)**	0.92 (+1.22%)	0.92 (+1.14%)
Naive Bayes	0.85	0.82 (-2.74%)	0.82 (-2.74%)	0.86 (+2.14%)	**0.87 (+3.11%)**	0.86 (+2.18%)
Random Forest	0.87	0.88 (+0.80%)	0.87 (+0.44%)	**0.91 (+4.20%)**	0.89 (+2.89%)	0.88 (+0.72%)
SVM	0.86	0.86 (+0.38%)	0.86 (+0.25%)	**0.92 (+6.85%)**	0.91 (+5.49%)	0.92 (+6.84%)
albacore (#1 in CBC)	0.95	–	–	**0.97 (+1.49%)**	0.95 (+0.27%)	–
zingel (#2 in CBC)	0.93	–	–	**0.95 (+1.86%)**	0.94 (+1.29%)	–

Table 2.5: Mean AUC scores and their relative changes (%) on $\mathcal{M}_{test}$ using different *oversampling* methods.

	Baseline	Synthesized Supervised Learning (SSL)				SMOTE
Algorithms	$\mathcal{M}_{train}$	$\mathcal{M}_{train} \cup \mathcal{C}$	$\mathcal{M}_{train} \cup \mathcal{S}$	$\mathcal{M}_{train} \cup \mathcal{A}^{\mathcal{M}}_{\text{VAE}}$	$\mathcal{M}_{train} \cup \mathcal{A}^{\mathcal{M}}_{\text{infoVAE}}$	$\mathcal{M}_{train} \cup \mathcal{O}^{\mathcal{M}}$
AdaBoost	0.68	0.69 (+1.12%)	0.69 (+1.12%)	**0.74 (+8.60%)**	0.71 (+4.74%)	0.72 (+5.80%)
Bagging Clf	0.67	0.67 (+0.09%)	0.67 (+0.42%)	**0.71 (+7.11%)**	0.68 (+2.84%)	0.67 (+0.93%)
Decision Tree	0.64	0.65 (+1.15%)	0.65 (+1.67%)	**0.67 (+3.66%)**	0.66 (+3.39%)	0.65 (+1.41%)
GradientBoosting	0.69	0.69 (+0.88%)	0.69 (+0.70%)	**0.74 (+7.93%)**	0.71 (+3.69%)	0.71 (+3.04%)
KNeighbors Clf	0.64	0.64 (+0.35%)	0.64 (+0.40%)	**0.69 (+8.11%)**	0.68 (+6.22%)	0.66 (+3.72%)
Logistic Regression	0.70	0.70 (+0.04%)	0.70 (+0.04%)	**0.74 (+6.02%)**	0.72 (+3.15%)	0.75 (+6.66%)
Naive Bayes	0.66	0.63 (-4.74%)	0.63 (-4.74%)	0.70 (+5.64%)	0.67 (+1.02%)	**0.72 (+7.96%)**
Random Forest	0.65	0.66 (+0.76%)	0.66 (+0.67%)	**0.71 (+9.42%)**	0.69 (+6.66%)	0.65 (+0.47%)
SVM	0.65	0.66 (+1.55%)	0.66 (+1.53%)	**0.75 (+14.56%)**	0.71 (+8.1%)	0.75 (+14.51%)
albacore (#1 in CBC)	0.71	–	–	**0.77 (+8.5%)**	0.75 (+6.0%)	–
zingel (#2 in CBC)	0.71	–	–	**0.76 (+6.9%)**	0.74 (+4.55%)	–

2.3.1 AQ1: Predictive Power of Synthetic Clickbaits

We examine how much generated synthetic clickbaits described in RQ1 can improve ML clickbait detection models. We name this process of generating a large amount of synthetic data to enhance supervised learning tasks as *Synthesized Supervised Learning (SSL)*. We further compare SSL with SMOTE and top-2 performed clickbait detectors from Clickbait Challenge 2017 (CBC).

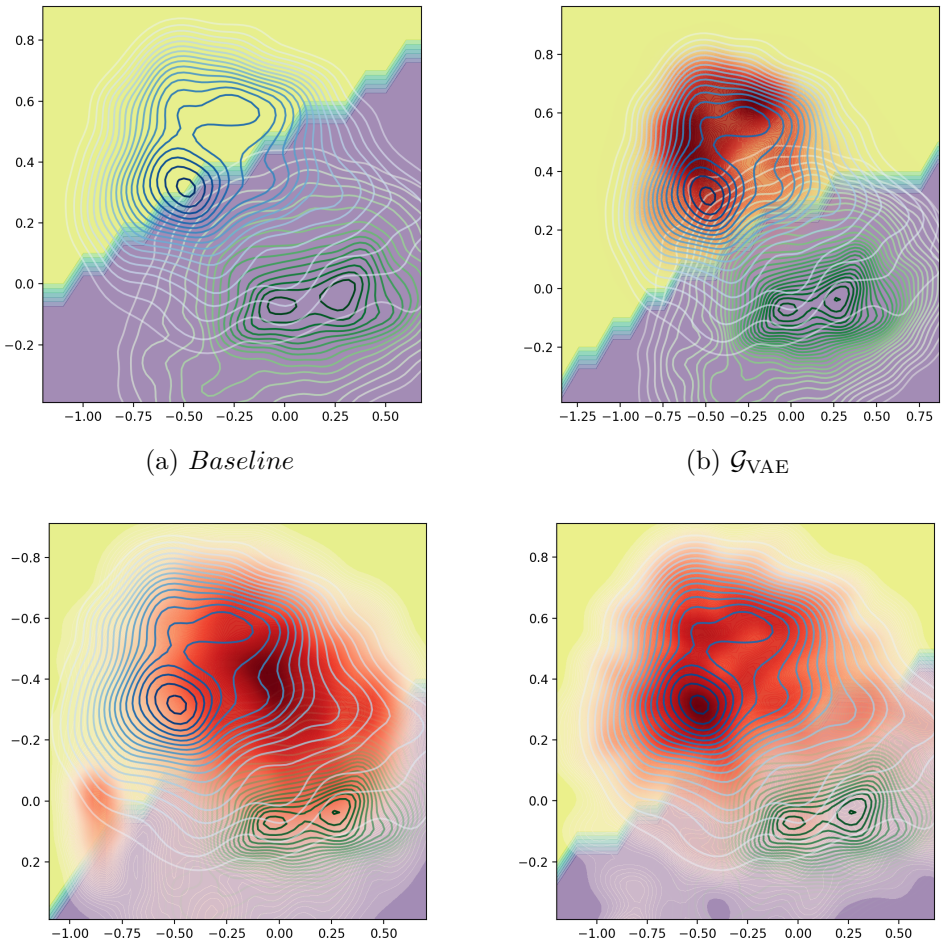
For classical ML algorithms, we use NLP-based features as input. Since this work does not aim to develop new features for predicting clickbaits, simply, we have selected several features from the literature that manifests different nuances in the use of language for writing headlines. They are selected because of their reported effectiveness in detecting clickbait or misleading headlines across different published works (e.g., [10, 12–14, 16]). Except for general POS-N-gram and Word-N-grams features, Table 2.6 lists all of the selected features. Being deep learning based, two top-performing models from CBC (albacore and zingel) automatically learn feature representation from a large amount of raw text data. Due to limited number of human-written synthetic clickbaits, we only examine this with machine-generated clickbaits. Since SMOTE cannot over-sample on raw data space, it is not applicable for the deep learning based detectors. We used open-source implementations published on the CBC website for two deep learning based models.

Table 2.6: *NLP Features Descriptions*

Type	Feature Description
Summary Statistics	Average word length, Stop-words ratio
	Counts of words, POS tags
	Length of the longest word
Sentiment	Intensity Score
Forward References	Pattern: (this/these/etc.) + Noun
Linguistic Patterns	Pattern: Number + Noun + That ?
	Pattern: Number + Noun + Verb ?
	Starting with a number, 5WH?
Informality	Flesch-Kincaid score
	Counts of Internet Slangs
Special Indicators	".", "!", "?", "@"; "http", "#"; "***"

We first constructed training and testing sets from $\mathcal{P}$ and $\mathcal{M}$ in the ratio of 3:1, resulting in $\mathcal{P}_{train}, \mathcal{P}_{test}$ and $\mathcal{M}_{train}, \mathcal{M}_{test}$ respectively. Then, to see if synthetic

`https://clickbait-challenge.org`



(a) *Baseline* (b) $\mathcal{G}_{\text{VAE}}$

(c) $\mathcal{G}_{\text{infoVAE}}$ (d) SMOTE

Figure 2.1: Decision boundary of a trained SVM classifier on $\mathcal{P}_{train}$ changed with and without different additional synthetic clickbaits. Blue contour, green contour, and red shade depict the density of positive, negative, and synthetic clickbaits, respectively.

clickbaits are useful to improve the detection of clickbaits, when they are added as additional labeled training samples, we first used *only* the positive training data in each of the datasets $\mathcal{P}_{train}$ and $\mathcal{M}_{train}$ to train generative models $\mathcal{G}_{\text{VAE}}$ and $\mathcal{G}_{\text{infoVAE}}$ as described in RQ1. The trained models were subsequently used to generate four respective synthetic datasets $\mathcal{A}^{\mathcal{P}}_{\text{VAE}}$. $\mathcal{A}^{\mathcal{M}}_{\text{VAE}}$, $\mathcal{A}^{\mathcal{P}}_{\text{infoVAE}}$, $\mathcal{A}^{\mathcal{M}}_{\text{infoVAE}}$. Next, we combined these

with the original training sets, $\mathcal{P}_{train}$ and $\mathcal{M}_{train}$, to train different predictive models, and tested against the original testing sets, $\mathcal{P}_{test}$ and $\mathcal{M}_{test}$, respectively.

Table 2.3 summarizes the datasets in this research.

Area-Under-the-Curve (AUC) is selected as the main evaluation measure for their robustness toward skewed labels distribution [18] of testing sets $\mathcal{P}_{test}$ and $\mathcal{M}_{test}$, where there exist 3–5 times more non-clickbaits than clickbaits. Such imbalanced distribution reflects real challenges, where we usually have many more non-clickbait than clickbait text.

For each algorithm, we reported relative changes of *AUC score* between models trained *with* and *without* additional synthetic clickbait datasets (baseline).

Tables 2.4 and 2.5 summarize experimental results. Numbers in **Bold** and underline indicate best of each row and column respectively. We showed that our framework helped improve on both algorithm-wise and dataset-wise. Particularly, from the same original training set, our approach of generating and using synthetic clickbaits was able to enhance the detection performance of both NLP-based and deep-learning based algorithms. Especially, $\mathcal{A}^{\mathcal{P}}_{\text{VAE}}$ and $\mathcal{A}^{\mathcal{M}}_{\text{infoVAE}}$ *consistently* improved *AUC scores* across all algorithms. Interestingly, performance of NLP-based algorithms with the proposed data-enhancement approach achieved comparable ($\mathcal{P}_{test}$) or even better ($\mathcal{M}_{test}$) than top-ranked deep learning models without the need of collecting any additional *real* data. Furthermore, clickbaits generated by generative models outperformed over-sampled data synthesized by SMOTE on all predictive algorithms for $\mathcal{P}_{test}$, and 8 out of 9 cases for $\mathcal{M}_{test}$. Noticeable, generative models outperformed SMOTE significantly on all of the ensemble-based classifiers across the testing sets on $\mathcal{M}_{test}$. The proposed framework can even further improve performance of top-ranked models from CBC by as much as 8.5%, achieving the best performance overall in both datasets.

In summary, we demonstrated that both models $\mathcal{G}_{\text{VAE}}$ and $\mathcal{G}_{\text{infoVAE}}$ could generate synthetic clickbaits (learned from training data) that, when added to training data, significantly improved domain-engineered predictive models. The fact that these NLP features have been built in many non-computational domains (e.g., journalism, communication, social science) illustrates that one may leverage the capability of generative models to model complex natural language distribution that reinforce our domain knowledge.

2.3.2 AQ2: NLP Feature Encapsulation

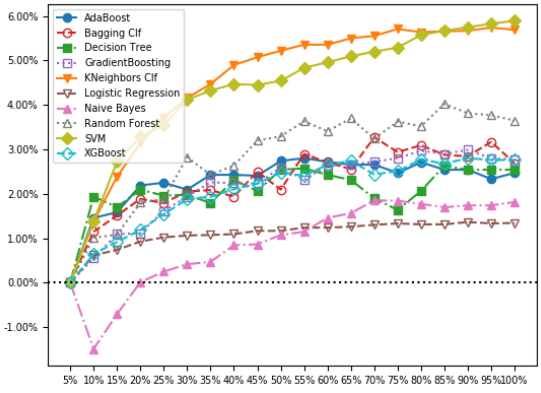
From the strong results reported in AQ1, we then examine whether synthetic clickbaits share the same distribution of NLP features as real positive clickbaits. We achieved this by both (1) visual examination and (2) analytic testing. For visual examination, we used $\mathcal{P}$ as an illustration. We first trained an Isomap dimension reduction model [19] on $\mathcal{P}_{train}$, and used the trained model to project the features extracted from $\mathcal{A}_{\text{VAE}^{\mathcal{P}}}$, $\mathcal{A}_{\text{infoVAE}^{\mathcal{P}}}$ and $\mathcal{O}^{\mathcal{P}}$ into a 2D feature space. Next, we trained an SVM classifier with new features and plotted its decision boundary between two class samples, resulting in Figure 2.1. Even though SMOTE over-sampled data directly on NLP features space, many new samples are mis-located in the original negative samples' area. In fact, without directly learning from feature set, NLP feature distributions of $\mathcal{A}_{\text{VAE}^{\mathcal{P}}}$ and $\mathcal{A}_{\text{infoVAE}^{\mathcal{P}}}$ are highly overlapped with the original positive samples. Especially, that of $\mathcal{A}_{\text{VAE}^{\mathcal{P}}}$ neatly concentrated around the center of original positive samples, while that of $\mathcal{A}_{\text{infoVAE}^{\mathcal{P}}}$ is located near the boundary between two original classes. Therefore, we can see that Figure 2.1 confirms predictive results of SVM on $\mathcal{P}_{test}$ in Table 2.4.

To analytically test, next, for each clickbait x_i in a synthetic dataset $\mathcal{Q}$, we extracted different NLP features listed in Table 2.6 and used a K Nearest-Neighbor (KNN) searching model to find its k nearest samples from the original training set $\mathcal{T}$ ($\text{NN}_{\mathcal{T}}(k, x_i)$) in the feature space, and calculated the ratio between the number of positive samples found over k. For each generated synthetic dataset, we averaged all the ratios to total $\mathcal{N}$ number of data points in $\mathcal{Q}$ to calculate a statistic:

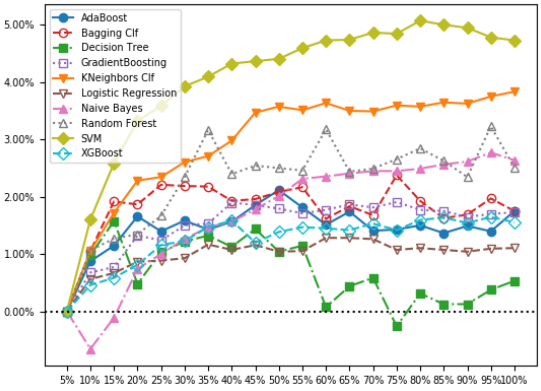
$$Overlap_{\text{NLP}}(k, \mathcal{Q}, \mathcal{T}) = \frac{1}{\mathcal{N}} \sum_{x_i \in \mathcal{Q}} \frac{|\text{NN}_{\mathcal{T}}(k, x_i) \cap \mathcal{T}_{pos}|}{k}. \tag{2.1}$$

This statistic captures on average how likely generated samples of a synthetic dataset $\mathcal{Q}$ will be close to original positive clickbaits in the feature space. We calculated such measure for each generated synthetic dataset and illustrated the result in Table 2.7. This result shows that $\mathcal{G}_{\text{VAE}}$-generated clickbaits are the one most overlapping with the original positive samples in the features space, which coincides with our visual examination in Figure 2.1. Overall, even though SMOTE directly generated data on the set of NLP predictive features, both $\mathcal{G}_{\text{VAE}}$ and $\mathcal{G}_{\text{infoVAE}}$ were better in capturing similar NLP structures of original clickbait data, resulting in better prediction of ML models than those NLP features, as reported

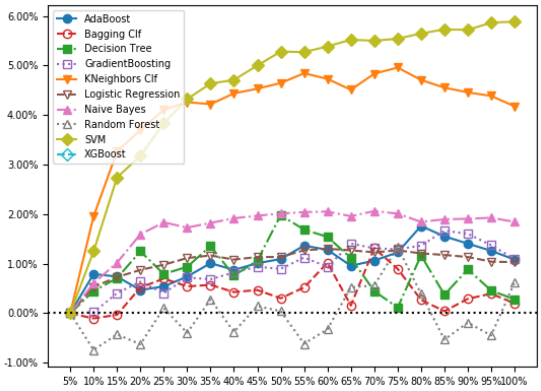
in AQ1.



(a) $\mathcal{G}_{\text{VAE}}$

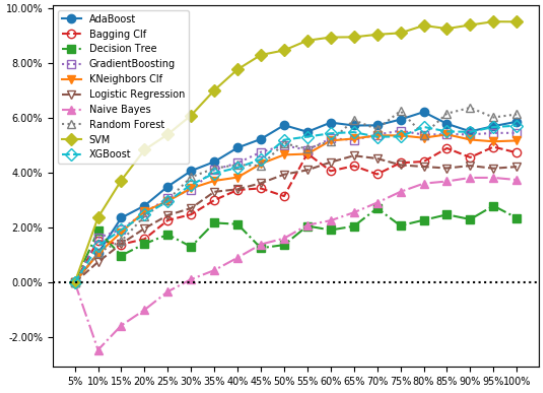


(b) $\mathcal{G}_{\text{infoVAE}}$

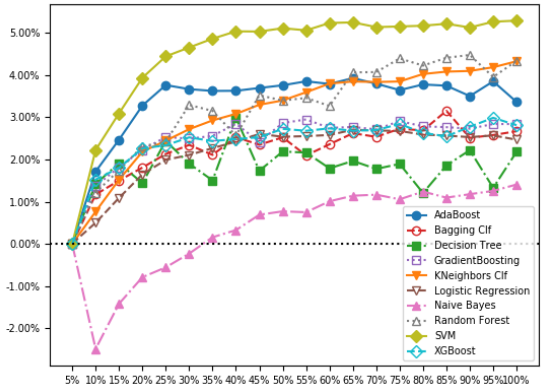


(c) SMOTE

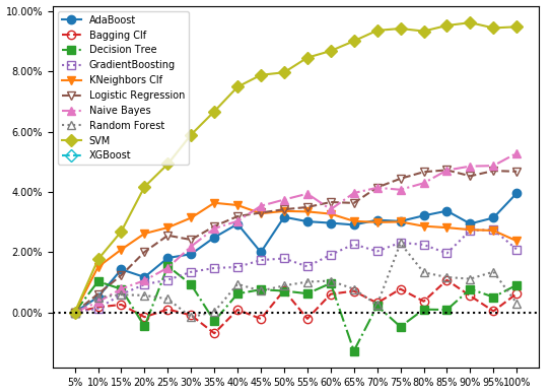
Figure 2.2: Proportion of additional synthetic clickbaits versus absolute AUC score improvement from baseline on $\mathcal{P}_{test}$



(a) $\mathcal{G}_{\text{VAE}}$



(b) $\mathcal{G}_{\text{infoVAE}}$



(c) SMOTE

Figure 2.3: Proportion of additional synthetic clickbaits versus absolute AUC score improvement from baseline on $\mathcal{M}_{test}$

2.3.3 AQ3: Robustness of Synthetic Clickbaits

In AQ2, we illustrated that different types of synthetic clickbaits improved clickbait detection models to different extents. In this section, we examine the robustness

Table 2.7: $Overlap_{\text{NLP}}$ score with $k = 5$ of synthetic datasets on $\mathcal{P}_{train}$ and $\mathcal{M}_{train}$

Statistic	$\mathcal{A}_{\text{VAE}}$	$\mathcal{A}_{\text{infoVAE}}$	$\mathcal{S}$	$\mathcal{C}$	SMOTE
$\mathcal{P}_{train}$	**0.7**	0.44	0.49	0.49	0.4
$\mathcal{M}_{train}$	**0.5**	0.44	0.38	0.38	0.35

of them.

Because of limited data samples generated by human users, we focus on the comparison between generative models and SMOTE. We only demonstrate on NLP-based models due to limited computational resources. We measure the robustness of an oversampling method by answering: (1) does a method improve predictive models with a small amount of additional generated samples? and (2) is such an improvement consistent as more data is added to the training set?

Figures 2.2 and 2.3 plot the relations between the amount of additional positive clickbait samples generated by $\mathcal{G}_{\text{VAE}}$, $\mathcal{G}_{\text{infoVAE}}$, SMOTE, and their improvements in absolute AUC for all of the examined algorithms.

Regarding the performance on $\mathcal{P}_{test}$, generative models only needed 20% of total additional training data until balanced to outperform the baseline across all of the algorithms, while SMOTE needed as much as 25% to achieve the same result. However, such performance of generative models showed a much larger improvement margin compared to SMOTE. Especially, as we add more synthetic clickbaits generated by $\mathcal{G}_{\text{VAE}}$, the improvement was more consistent, showing smoother improvement lines in absolute AUC, compared to the cases of $\mathcal{G}_{\text{infoVAE}}$ and SMOTE. The same outcome was also observed in the case of $\mathcal{M}_{test}$. In fact, only 30% of total clickbaits generated by $\mathcal{G}_{\text{VAE}}$ was needed to outperform 100% of data sampled by SMOTE (balanced training set) in most algorithms in both datasets.

Overall, generative algorithms generated more robust synthetic clickbaits than SMOTE, showing consistent and continuous improvements while adding more training data.

2.3.4 AQ4: Interpretability of Synthetic Clickbaits

Humans and deep generative models, $\mathcal{G}_{\text{VAE}}$ and $\mathcal{G}_{\text{infoVAE}}$, clearly have advantages over SMOTE in terms of interpretability. Algorithms-wise, SMOTE samples data only on feature space, i.e., numerical features extracted from text domain, the

results of which cannot be converted back to the data space, i.e., natural text. However, $\mathcal{G}_{\text{VAE}}$ and $\mathcal{G}_{\text{infoVAE}}$ learn and generate natural sentences that are interpretable to humans (e.g., Table 2.1). This shows that while the samples generated by SMOTE are task-independent, i.e., they are represented only on a pre-defined set of features, the sentences produced by generated models can transfer to other tasks or domains such as misinformation analysis.

2.4 RQ3: Differentiate Clickbaits per Sources

Next, we ask if entity-cross differences among synthetic clickbaits are consistent and identifiable by ML models. This type of study can be also useful in a security scenario–e.g., malicious publishers take an advantage of different entities to generate clickbaits to propagate low-quality news content, or to attract more traffic. A demo system such as Click-O-Tron and Link Bait Title Generator illustrates the possibility of such an attack scenario to mass-generate clickbaity headlines with malevolent intents.

From the synthetic clickbait datasets in Table 2.3, we aim to achieve the following specific objectives:

Obj1 Can we distinguish among clickbaits in $\mathcal{P}$, $\mathcal{M}$, $\mathcal{C}$, $\mathcal{S}$, $\mathcal{A}$?

Obj2 Can we distinguish among clickbaits by trained writers ($\mathcal{P} \cup \mathcal{S}$), general public ($\mathcal{M} \cup \mathcal{C}$), and machine ($\mathcal{A}$)?

Obj3 Can we distinguish clickbaits by humans ($\mathcal{P} \cup \mathcal{M} \cup \mathcal{C} \cup \mathcal{S}$) vs. machine ($\mathcal{A}$)?

These tasks can be modeled as three different multinomial classification problems. Since the nature of these tasks is similar to the ones in the previous section, we re-use some of the introduced algorithms by changing the ground-truth labels accordingly. From analysis in section RQ2, we select $\mathcal{A}_{\text{VAE}}$ as the representative synthetic clickbait set generated by machine $\mathcal{A}$ because it better captures characteristics of real clickbaits than $\mathcal{A}_{\text{infoVAE}}$.

Table 2.8 summarizes the experimental results, where baselines (i.e., random guess) have accuracies of 20% for Obj1 in differentiating clickbaits of five different

http://clickotron.com/
http://www.contentrow.com/tools/link-bait-title-generator

Table 2.8: Clickbaits' Source Verification Benchmark

Alg	Obj1		Obj2		Obj3	
	Acc	F1_avg	Acc	F1_avg	Acc	F1_avg
LogReg	0.54	0.54	0.59	**0.58**	0.61	0.62
NBayes	0.52	0.50	0.56	0.54	0.57	0.59
DTree	0.47	0.47	0.50	0.50	0.61	0.61
RForest	0.54	0.53	0.56	0.55	0.67	**0.65**
XGBoost	0.58	0.52	0.61	0.55	0.70	0.59
AdaBoost	0.55	0.50	0.59	0.55	**0.70**	0.62
SVM	0.57	**0.56**	0.60	0.58	0.61	0.57
GradBoost	**0.59**	0.55	**0.61**	0.56	0.70	0.60
Bagging	0.54	0.53	0.57	0.56	0.66	0.64
KNeighbor	0.51	0.49	0.53	0.51	0.66	0.64

Table 2.9: Top distinguishing features

Top	Obj1	Obj2	Obj3
1	avg word length	# of end mark	# of end mark
2	# of words	avg word length	avg word length
3	% of stop words	# of words	% of stop words
4	# of end mark	% of stop words	# of POS tags
5	# of POS tags	start with number?	# of JJ-NN

entities, 33.3% for Obj2 in distinguishing between trained writers, general public and machine, and 50% for Obj3 in classifying between human-written and machine-generated clickbaits. Note that all three objectives can be achieved with accuracy as high as 59%, 61% and 70%, all of them considerably higher than those of baselines.

Overall, while it is challenging to differentiate clickbaits written by different sources, we achieve reasonable results on *accuracy* and *average F1 score* measures. It is especially encouraging that we can distinguish clickbaits generated by machine from those generated by human with as high as 65% in averaged F1 score. This further demonstrates the utility of our synthetic clickbaits in developing models that have strong potential for empirical use.

Table 2.9 illustrates top predictive features resulting from the Gradient Boosting classifier trained for the three objectives. Since we are grouping some entities in Obj1 to examine Obj2 and Obj3, the result shows many repeated top features across all three tasks. Among the five groups of clickbait headlines, average word length is the most distinguishable feature. Journalism students use longer words in

their clickbaits compared to other entities. We find that crowdworkers and students use significantly lower number of Wh-determiners (which, that, etc. as determiners) in their headlines compared to other sources. Also, professional writers use personal pronouns (I, you, he, she, etc.) much more often than other entities. We also see differences in other writing strategies among the five entities. Specifically, professional writers are more likely to start their clickbaits with numbers (e.g. "20 things to do before 20"), and media users are more more likely to use question and exclamation marks and more than single sentence in their headlines.

Despite the fact that the generative algorithms can be biased towards the type of clickbaits having the majority of training samples (professional writers), the fact that it still generates clickbaits that simulate human behaviors, which eventually makes it very challenging for us to differentiate, is intriguing. As indicated in Table 2.9, many of the features that best distinguish the two groups of clickbaits are counts of various POS tags and their combinations. The generative algorithm's strategy might have been learning to replicate different collocations from human-written clickbaits. Moreover, it also learns the relative position of those phrases.

2.5 Discussion

2.5.1 Utility of Synthetic Text.

There have been prior works in generating natural-looking, realistic, and human-readable synthetic text (e.g., [**?**, 20, 21]). However, few of them have explored the characteristics and utility of synthetic text for downstream machine learning tasks such as prediction and clustering. In fact, to perform well in these machine learning tasks, synthetic text does not have to be realistic and coherent, but must capture certain characteristics or domain knowledge of original text. By using clickbait domain as a case study, in this work, we have demonstrated that synthetic text (generated by diverse methods) can help improve classification tasks and introduce insights into domain specific problems. Generalizing the findings to other domains and applications will be our future work.

2.5.2 Implications for Combating Misinformation.

Our findings highlight the promise in using generative algorithms to detect misinformation (spam, fake news, etc.), a domain that usually lacks high-quality labeled data. RQ2 illustrates that the aggregation of synthetic clickbaity text by both humans and machines can be beneficial to improve clickbait detection accuracy by as much as 14.5% in *AUC scores*. Moreover, machine-generated clickbaits (RQ1) can be used to develop a defense mechanism to battle against mass propagation of false information initiated by malicious bots in social networks, which would help human fact-checkers focus more on detecting intentional misinformation.

This chapter suggests features that are useful not only for developing algorithms that both effectively detect and discriminate various types of clickbaits, but also for training humans to become more aware and sensitive to potential misinformation by attaching a source label to flagged clickbaits. The outcomes also provide insights on the potential presentation of clickbaity headlines. To illustrate, RQ1 shows that formally-trained journalism students often present clickbaity headlines with political context even for non-political target content, while such behaviors are not observed among social media users. Similar behavior and its influence has been studied in detail by [22, 23]

Similar behavior was also observed in the case of "Syrian social bot" on Twitter, which spread Syrian civil war related hashtags with contents that are not war related [22]. Interestingly, Geer and Kha [23] show that alternations in topics of headlines actually result in more influence than the slant of a specific issue. This further emphasizes that fact checkers should pay attention to *the source of news*, as different publishers might target different "cultures of communications such as readers' cognitive styles, reading proficiency, and interest" [24]. This also implies that clickbaits or rumor detection algorithms that are customized for a specific topic (e.g., political election) should take the associated article's context into consideration.

2.6 Related Work

2.6.1 Collecting, Generating Clickbaits.

Researchers have attempted to collect and build labeled clickbait datasets, by using the following approach. First, recognizing that certain online news media outlets frequently use catchy headlines, researchers collect headlines from such sites as candidate clickbaits. Second, as the definition of clickbaits is often fuzzy and subjective, researchers tend to rely on the voted labels of candidate clickbaits from human judges or crowdworkers (e.g. [12, 14, 25]).

However, the generation aspect of clickbaits was never a focus in these works. A recent attempt to "generate" clickbaits is found in Click-O-Tron that trains the RNN with millions of articles from sites such as BuzzFeed, Huffington Post, and Upworthy. Algorithmically, this line of work can be derived from the task of language modeling and text generation in AI. There has been considerable progress in generating realistic text, either in randomized (e.g., [26]) or controllable fashion (e.g., [**?**, 21]). Leveraging these developments, this work adopts VAE-based generative models [15, 17] to demonstrate the generation of realistic clickbaits. Unlike these works, however, in Section RQ1, we also illustrate experimental designs to generate clickbaits by different human creators (e.g., crowdworkers and journalism students). Throughout this chapter, we adopt the two public datasets curated by [14] collected from *Professional* publishing websites, and by [12] collected from *Social Media Twitter* as datasets $\mathcal{P}$ and $\mathcal{M}$ respectively.

2.6.2 Postmortem: Detecting Clickbaits.

Clickbait detection has attracted increasing attention in recent years. Most of existing clickbait detection approaches explore engineering features in a supervised ML framework (e.g. [14, 16, 27]). More recently, researchers have employed the deep neural framework to automatically learn latent features from clickbaits (e.g. [8, 25, 28]).

Many of these attempts focus on extracting different features and building a predictive model to approach the problem, yet they are bounded by the availabil-

http://clickotron.com/

ity and quality of existing labeled training datasets. Note that this chapter is *not* aiming at directly comparing against these existing works. Rather, our ideas in RQ1 explore the potentials of generating synthetic clickbaits and utilizing them in improving detection models further.

2.7 Limitation and Future Direction.

A more thorough study of text generation as an oversampling method by other models (e.g., GAN-based) is of future interest. Insights gained therefrom will enable us to frame a better oversampling method that can better generate useful samples. Even though we are not trying to generate realistic text clickbaits, we plan to carry out a field survey to analyze and compare how users would perceive and react to synthetic clickbaits generated by different entities, and to answer the question: "how clickbaity are they?" Finally, we plan to apply the framework in other domains where collecting training data is either challenging or limited (e.g., rumor detection, writing-based Alzheimer detection).

2.8 Conclusion

We explored the utility of synthetically generated text in the context of clickbaits, and demonstrated that synthetic clickbaits can be useful as additional labeled training samples to train regular ML models to detect clickbaits better, by as high as 14.5% in AUC. We showed that VAE-based generative algorithms can generate high quality text that captures the most similar NLP feature distribution as the real ones among all synthetic sources. Even though such an overlap in NLP feature distribution does not directly make synthetic clickbaits as meaningful as real clickbaits, the outcomes demonstrated a promising track in using machines to generate realistic text in general. This framework can, thus, present a novel direction toward solving the problem of insufficient training data in supervised learning.

Chapter 3
Learning with Transparency: Counterfactual Explanation for Neural Network Classifiers

3.1 Background

Tabular data is one of the most commonly used data formats. Even though tabular data receives far less attention than computer vision and NLP data in neural networks literature, recent efforts (e.g., [29–32]) have shown that neural networks, deep learning in particular, can also achieve superior performance on this type of data. Yet, there is still a lack of interpretability that results in the distrust of neural networks trained on general tabular data domains. This obstructs the wide adoption of such models in many high-stakes scenarios in which tabular data is prominent–e.g., healthcare [33, 34], finance [35, 36], social science [37, 38], and cybersecurity [39, 40]. Moreover, the majority of explanation algorithms (e.g., [**?**, 4, 41–45]) are designed for models trained on images or texts, while insufficient efforts have been made to explain the prediction results of neural models that take tabular data formats as input. Furthermore, most of the previous explanation approaches are geared for professional users such as ML researchers and developers rather than lay users and ML consumers.

This situation calls for a novel approach to provide end-users with the intuitive explanation of neural networks trained on tabular data. However, developing such an approach poses several challenges. **Challenges.** *First*, tabular data used in

Table 3.1: Examples of original samples $\boldsymbol{x}_i$ and contrastive samples $\tilde{\boldsymbol{x}}_i$ on *spam* dataset. $\tilde{x}_i$ only differs from $\boldsymbol{x}_i$ on *a few* features.

Feat	freq_now	freq_credit	freq_!!!	freq_!	class
$\boldsymbol{x}_1$	0.1	0.0	0.0	0.0	Ham
$\tilde{\mathbf{x}}_\mathbf{1}$	0.1	0.0	**0.3**	**0.453**	**Spam**
Feat	freq_you	freq_direct	avg_longest_capital		class
$\boldsymbol{x}_2$	0.68	0.34	158.0		Spam
$\tilde{\mathbf{x}}_2$	0.68	0.34	**1.0**		**Ham**

neural network models sometimes have high-dimensional inter-correlated features. Therefore, presenting feature importance scores for top-k or all features (e.g., [4]) can induce both information overload and redundancy, causing confusion to end-users.

In fact, for a data instance, a complex model can focus on just a few key features in making its prediction. To illustrate, Table 3.1 shows that for two emails $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the model can focus on two different sets of features, $freq\_!$, $freq\_!!!$ or $avg\_longest\_capital$, respectively, to predict if an email is a spam or ham. While explanation constructed **only** from these features is much more concise, providing both $freq\_!$ and $freq\_!!!$ (frequency of "!" and "!!!" within an email content) in the first example produces redundancy. In this case, we also want to replace $freq\_!$ with another key feature to make the explanation more informative. Thus, we need to find a subset of instance-dependent key features that are both *concise* and *informative* to explain the model's prediction.

Second, for images or texts, highlighting a patch of an image (e.g., [41, 44, 45]) or a phrase of a sentence (e.g., [**?**]) usually gives a clear understanding of what a model is focusing on and why a model gives such prediction.

However, in tabular data, such visualization does not provide much insight into the chosen model. For instance, in the second example in Table 3.1, the model predicts $\boldsymbol{x}_2$ as spam and the important feature used by the model is $avg\_longest\_capital$. However, simply providing this feature to end-users does not give an easy-to-understand explanation. Since we often justify our decision verbally [46], in this case, an explanation written in text can help end-users understand the prediction better.

Third, approximating the decision boundaries does not necessarily provide a clear understanding on the decision-making of a model to end-users, who usually

lack ML background. Instead, such lay users are usually more interested in the **contrastive explanation**, i.e., why X rather than Y. For example, Table 3.1 shows in the second example that "had *avg_longest_capital* (i.e., the average length of the longest capitalized words) been about 150 characters shorter, the email would have been classified as **ham rather than spam**". Hence, we need to come up with a new explanation model such as *contrastive explanation* to better explain a model's prediction to lay end-users.

Overview. To sum up, the effort towards generating an explanation that is easy for end-users to understand is challenging, yet also in great demand. Therefore, we propose a novel algorithm, GRACE (*GeneRAting Constrastive samplEs*), which generates and provides end-users with intuitive and informative explanations for neural networks trained on general tabular data. Inspired from Database (DB) literature [47–49], GRACE borrows the idea of "explanation by intervention" from causality [**?**, 50] to come up with contrastive explanation–i.e., why a prediction is classified as X rather than Y. Specifically, for each prediction instance, GRACE generates an explainable sample and its contrastive label by selecting and modifying a few instance-dependent key features under both *fidelity*, *conciseness* and *informativeness* constraints. Then, GRACE aims to provide a friendly text explanation of *why X rather than Y* based on the newly generated sample.

The main contributions of the chapter are:

- We introduce an explanation concept for ML by marrying "contrastive explanation" and "explanation by intervention", then extend it to a novel problem of generating contrastive sample to explain why a neural network model predicts X rather than Y for data instances of tabular format;
- We develop a novel framework, GRACE, which finds key features of a sample, generates contrastive sample based on these features, and provides an explanation text on why the given model predicts X rather than Y with the generated sample; and
- We conduct extensive experiments using eleven real-world datasets to demonstrate the quality of generated contrastive samples and the effectiveness of the final explanation. Our user-studies show that our generated explanation texts are more intuitive and easy-to-understand, and enables lay users to make as much as 60% more accurate post-explanation decisions than that of LIME.

3.2 The Explanation Model

3.2.1 Contrastive Explanation

Understanding the answer to the question "Why?" is crucial in many practical settings, e.g., in determining why a patient is diagnosed as benign, why a banking customer should be approved for a housing loan, etc. The answers to these "Why?" questions can be really answered by studying causality, which depicts the relationship between an event and an outcome. The event is a cause if the outcome is the consequence of the event ([48, 50]). However, causality can only be established under a controlled environment, in which one alters a single input while keeping others constant, and observes the change of the output. Bringing causality into data-based studies such as DB or ML is a very challenging task since causality cannot be achieved by using data alone ([48]). As the first step to understand causality in data-intensive applications, DB and ML researchers have tried to lower the bar of explanation, aiming to find the subset of variables that are best correlated with the output. Specifically, DB literature aims to provide explanations for a complex query's outputs given all tuples stored in a database, while ML researcher is keen on explaining the predictions of learned, complex models.

3.2.2 Explanation by Intervention

By borrowing the notion of *intervention* from causality literature, in particular, DB researchers have come up with a practical way of explaining the results of a database query by searching for an explainable predicate $\mathcal{P}$. Specifically, $\mathcal{P}$ is an explanation of outputs $\mathbf{X}$ if the removal of tuples satisfying predicate $\mathcal{P}$ also changes $\mathbf{X}$ while keeping other tuples unchanged ([47–49]). Similarly, by utilizing the same perspective, we want to formulate a definition of *explanation by intervention* for ML models at instance-level as follows.

Definition 1 (Contrastive Explanation (in ML) by Intervention) *A predicate $\mathcal{P}$ of subset of features is an explanation of a prediction outcome $\mathbf{X}$, if changes of features satisfying the predicate $\mathcal{P}$ also changes the prediction outcome to $\mathbf{Y}(\neq \mathbf{X})$,* ***while keeping other features unchanged****.*

For example, possible predicates to explain a spam detector are shown in Table 3.1. Particularly, predicate $\mathcal{P}_2$: "avg_longest_capital = 1.0" explains why sample $\boldsymbol{x}_2$ is classified as spam rather than ham. Given a prediction of a neural network model on an input, there will be possibly many predicates $\mathcal{P}$ satisfying Def. 1. Hence, it is necessary to have a measure to describe and compare how much influence predicate(s) $\mathcal{P}$ have on the final explanation. Following the related literature of explanation from the DB domain [47], we also formally define a scoring function $\mathit{infl}_\lambda(\mathcal{P})$ as the measure on the influence of $\mathcal{P}$ on the explanation with a tolerance level λ as.

Definition 2 (Influence Scoring Function)

$$\mathit{infl}_\lambda(\mathcal{P}) = \frac{\mathbb{1}(\mathbf{Y} \neq \mathbf{X})}{(\textit{Number of features in } \mathcal{P})^\lambda} \tag{3.1}$$

where $\mathbb{1}(\cdot)$ *is an indicator function,* $\mathbf{X}$ *and* $\mathbf{Y}$ *are predicted labels before and after intervention, respectively.*

The larger the score is, the more influential $\mathcal{P}$ has on the explanation. Hence, $\lambda = 0$ would imply infinite tolerance on the number of features in $\mathcal{P}$, $\lambda > 0$ would prefer a small size of $\mathcal{P}$ and $\lambda < 0$ would prefer a large size of $\mathcal{P}$. In practice, $\lambda > 0$ is preferable because a predicate $\mathcal{P}$ containing too many features would adversely affect the comprehension of the explanation. For example, $\mathit{infl}_1(\mathcal{P}_2) = 1.0$

3.2.3 From Intervention to Generation

Searching for $\mathcal{P}$ is a non-trivial problem. From Def. 1, we want to approach this problem from a generation perspective. Particularly, given an arbitrary sample classified as $\mathbf{X}$ by a neural network, we want to intervene and modify a small subset of its features to generate a new sample that crosses the decision boundary of the model to class $\mathbf{Y}$. This subset of features and their new values will result in a predicate $\mathcal{P}$. This newly generated sample will help answer the question "Why X rather than Y?". To illustrate, Table 3.1 shows that $\tilde{\mathbf{x}}_\mathbf{2}$ is generated samples that correspond to predicate $\mathcal{P}_2$: "avg_longest_capital = 1.0". Using $\mathcal{P}_2$, we can generate an explanation text to present to the users such as "Had the average

length of the longest capitalized words been 1.0, the message would have been classified as **ham rather than spam**".

3.3 Objective Function

$f(\cdot)$ be a neural network model that we aim to give instance-level explanation. Denote $\mathcal{X} \in \mathbb{R}^{N\times M} = \{x_1, x_2, ..x_n\}$, $\mathcal{Y} = \{y_1, y_2, ..y_n\}$ as the features and ground-truth labels of data on which $f(\boldsymbol{x})$ is trained, where N, M is the number of samples and features, respectively. $\mathcal{X}^i$ and $\mathcal{X}^j$ are the i-th and j-th feature, respectively, in features set $\mathcal{X}$. $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$ are the i-th and j-th feature of $\boldsymbol{x}$, respectively. First, we want to generate samples that are contrastive. We define such characteristic as follows.

Definition 3 (Contrastive Sample) *Given an arbitrary $\boldsymbol{x} \in \mathcal{X}$, $\mathcal{X} \in \mathbb{R}^{N\times M}$ and neural network model $f(\cdot)$, $\tilde{\mathbf{x}}$ is called contrastive or contrastive sample of $\boldsymbol{x}$ when:*

$$\min_{\tilde{\boldsymbol{x}}} dist(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \quad s.t. \quad \text{argmax}(f(\boldsymbol{x})) \neq \text{argmax}(f(\tilde{\boldsymbol{x}})) \tag{3.2}$$

Then, formally, we study the following problem:

Problem: Given $\boldsymbol{x}$ and neural network model $f(\cdot)$, our goal is to generate new contrastive sample $\tilde{\mathbf{x}}$ to provide concise and informative explanation for the prediction $f(\boldsymbol{x})$.

Existing works on adversarial example generation [51,52] usually define $dist(\tilde{\boldsymbol{x}}, \boldsymbol{x})$ as $\|\boldsymbol{x} - \tilde{\mathbf{x}}\|_2^2$, which allows all features to be changed to generate $\tilde{\boldsymbol{x}}$. Though such approaches can generate realistic labeled contrastive samples, they are not appropriate for generating instance-level explanation that are easy to understand because all features are changed.

Instead, we aim to generate $\tilde{\mathbf{x}}$ labeled $\tilde{\mathbf{y}}$ such that it is minimally different from original input $\boldsymbol{x}$ in terms of only a *few important features* instead of *all features*. Specifically, we desire to explain "Why X rather than Y?" by presenting a *concise* explanation in which only a few features are corrected, e.g., the first example of Table 3.1 shows if only frequency of '!' is increased to 45.3%, *while keeping other features unchanged*, the email will be classified as "Spam" rather than

"Ham". Hence, the less the number of features need to change from $\boldsymbol{x}$ to generate $\tilde{\mathbf{x}}$, the more "concise" the explanation becomes. To achieve this goal, we add the constraint as

$$|\mathcal{S}| \leq K \tag{3.3}$$

where $\mathcal{S}$ is the feature set of $\boldsymbol{x}$ that are perturbed to generate $\tilde{\boldsymbol{x}}$, hence making $|\mathcal{S}|$ the number of features changed, i.e.,

$$|\mathcal{S}| = \sum_{m=1}^{M} \mathbb{1}(\mathbf{x^m} \neq \tilde{\mathbf{x}}^{\mathbf{m}}) \tag{3.4}$$

We not only want to change a minimum number of features, but also want those features to be *informative*. For example, the explanation "Had the frequency of '!' and '!!!' is more than 0.3, the email would be classified as spam rather than ham" is not as informative as "Had the frequency of '!' and 'wonder' is more than 0.3, the email would be classified as spam rather than ham." Hence, we want $\mathcal{S}$ to contain a list of perturbed features such that any pairwise mutual information among them is within an upper-bound γ. Thus, we add the constraint:

$$\mathrm{SU}(\mathcal{X}^i, \mathcal{X}^j) \leq \gamma \quad \forall i, j \in \mathcal{S} \tag{3.5}$$

where $\mathrm{SU}(\cdot)$ is *Symmetrical Uncertainty* function, a normalized form of mutual information, to be introduced in Section 3.4.2.

Finally, we also need to ensure that the final predicate $\mathcal{P}$ is realistic. For example, the age feature should be a positive integer). Therefore, we want to generate $\tilde{\mathbf{x}}$ such that it conforms to features domain constraints of the dataset:

$$\tilde{\mathbf{x}} \in dom(\mathcal{X}) \tag{3.6}$$

Newly introduced constraints are novel from previous adversarial literature, which focus more on minimizing the difference $\|\boldsymbol{x} - \tilde{\mathbf{x}}\|_p$. However, minimizing such a distance alone will not necessarily make $\tilde{\mathbf{x}}$ more self-explanatory to users. Instead, we propose that as long as the constraints on the maximum number of perturbed features, i.e., Eq. (3.3), their entropy, i.e., Eq. (3.5), and domain, i.e., Eq. (3.6), is satisfied, we can generate more concise and informative explainable

contrastive samples. From the above analysis, we formalize the objective function as follows.

Objective Function: Given $\boldsymbol{x}$, hyperparameter K, γ, our goal is to generate new contrastive sample $\tilde{\mathbf{x}}$ to explain the prediction $f(\boldsymbol{x})$ by solving the objective function:

$$\begin{aligned} \min_{\tilde{\boldsymbol{x}}} \quad & dist(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \\ s.t. \quad & \text{argmax}(f(\boldsymbol{x})) \neq \text{argmax}(f(\tilde{\boldsymbol{x}})), \quad |\mathcal{S}| \leq K \\ & \text{SU}(\mathcal{X}^i, \mathcal{X}^j) \leq \gamma \quad \forall i, j \in \mathcal{S}, \quad \tilde{\mathbf{x}} \in dom(\mathcal{X}) \end{aligned} \tag{3.7}$$

3.4 GRACE: Generating Interventive Contrastive Samples for Model Explanation

This section describes how to solve the objective function and gives details GRACE. Figure 3.1 gives an illustration of the framework. It consists of three steps: (i) entropy-based forward features ranking, which aims at finding instance-dependent features satisfying the constraint; (ii) generate contrastive samples with the selected features; and (iii) create an explanation text based on generated sample $\tilde{\boldsymbol{x}}$. Alg. 1 describes GRACE algorithm.

3.4.1 Contrastive Sample Generation Algorithm

Before introducing how to obtain a list of potential features to perturb, in this section, we first describe our contrastive sample generation algorithm by assuming that the ordered feature list $\mathcal{U}^*$ is given. To solve $\tilde{\boldsymbol{x}}$ such that Eq. (3.2) is satisfied, we can continuously perturb $\tilde{\boldsymbol{x}}$ by projecting itself on the decision hyperplane separating it with the nearest contrastive class v. Particularly, at each time-step i, we project $\tilde{\boldsymbol{x}}$ with orthogonal projection vector $\mathbf{r}_v$:

$$\mathbf{r}_v = \frac{|f_v(\tilde{\boldsymbol{x}}_{i-1}) - f_C(\boldsymbol{x})|}{\|\nabla f_v(\tilde{\boldsymbol{x}}_{i-1}) - \nabla f_C(\boldsymbol{x})\|_2^2} \left(\nabla f_v(\tilde{\boldsymbol{x}}_{i-1}) - \nabla f_C(\boldsymbol{x})\right) \tag{3.8}$$

Algorithm 1 GRACE

Input: $f, \boldsymbol{x}$, K, γ, $\mathcal{X}$
Output: $\tilde{\boldsymbol{x}}$, $\tilde{\mathbf{y}}$

1: *Initialize:* $\tilde{\boldsymbol{x}} \longleftarrow \boldsymbol{x}$, $j \longleftarrow 1$
2: $\mathcal{U} \longleftarrow W_{Gradient}(f, \boldsymbol{x})$ **OR** $W_{Local}(f, \boldsymbol{x})$
3: $\mathcal{U}^* \longleftarrow W_{Entropy}(\mathcal{X}, \mathcal{U}, \gamma)$
4: **while** $\text{argmax}(f(\tilde{\boldsymbol{x}})) = \text{argmax}(f(\boldsymbol{x}))$ **AND** $k \leq K$ **do**
5: $\quad \tilde{\boldsymbol{x}} \longleftarrow$ GenerateContrastiveSample$(f, \boldsymbol{x}, \mathcal{U}^*[:k])$
6: $\quad k \longleftarrow k + 1$
7: **end while**
8: $\tilde{\mathbf{y}} \longleftarrow \text{argmax}(f(\tilde{\boldsymbol{x}}))$
9: **Return** $\tilde{\boldsymbol{x}}$, $\tilde{\mathbf{y}}$

where $f_v(\tilde{\boldsymbol{x}}_{i-1})$ is the confidence of f on $\tilde{\boldsymbol{x}}_{i-1}$ being classified as class v. $C \longleftarrow \text{argmax}(f(\boldsymbol{x}))$ is the current prediction label, and contrastive class $\boldsymbol{v}$ can be inferred with Alg. 2, Ln. 2. Intuitively, $\boldsymbol{v}$ is the contrastive class across the *closest* hyperplane of the decision boundary from $\boldsymbol{x}$.

Algorithm 2 GenerateContrastiveSample

Input: $f, \boldsymbol{x}$, $\mathcal{S}$
Output: $\tilde{\boldsymbol{x}}$

1: *Initialize:* $\tilde{\mathbf{x}}_{\mathbf{0}} \longleftarrow \boldsymbol{x}$, $i \longleftarrow 0$, $C \longleftarrow \text{argmax}(f(\boldsymbol{x}))$
2: $v \longleftarrow \text{argmin}_{c \neq C} \frac{|f_c(\boldsymbol{x}) - f_C(\boldsymbol{x})|}{||\nabla f_c(\boldsymbol{x}) - \nabla f_C(\boldsymbol{x})||_2}$
3: **while** $\text{argmax}(f(\tilde{\boldsymbol{x}}_i)) = \text{argmax}(f(\boldsymbol{x}))$ **AND** $i <$ STEPS **do**
4: $\quad \mathbf{r}_v = \frac{|f_v(\tilde{\boldsymbol{x}}_{i-1}) - f_C(\boldsymbol{x})|}{\|\nabla f_v(\tilde{\boldsymbol{x}}_{i-1}) - \nabla f_C(\boldsymbol{x})\|_2^2} (\nabla f_v(\tilde{\boldsymbol{x}}_{i-1}) - \nabla f_C(\boldsymbol{x}))$
5: $\quad \tilde{\boldsymbol{x}}_{i+1}[\mathcal{S}] \longleftarrow \tilde{\boldsymbol{x}}_i[\mathcal{S}] + \boldsymbol{r}_v[\mathcal{S}]$
6: $\quad \tilde{\boldsymbol{x}}_{i+1} \longleftarrow P(\tilde{\boldsymbol{x}}_{i+1}, dom(\mathcal{X}))$
7: $\quad i \longleftarrow i + 1$
8: **end while**
9: **Return** $\tilde{\boldsymbol{x}}$

To address constraint Eq. (3.3), instead of perturbing all of the features, we update $\tilde{\boldsymbol{x}}$ only on the first $k \leq K$ features from an ordered list $\mathcal{U}^*$, which will be introduced later, at each time step i until it crosses the decision boundary:

$$\mathcal{S} \longleftarrow \mathcal{U}^*[:k], \quad \tilde{\boldsymbol{x}}_i[\mathcal{S}] \longleftarrow \tilde{\boldsymbol{x}}_{i-1}[\mathcal{S}] + \mathbf{r}_v[\mathcal{S}] \tag{3.9}$$

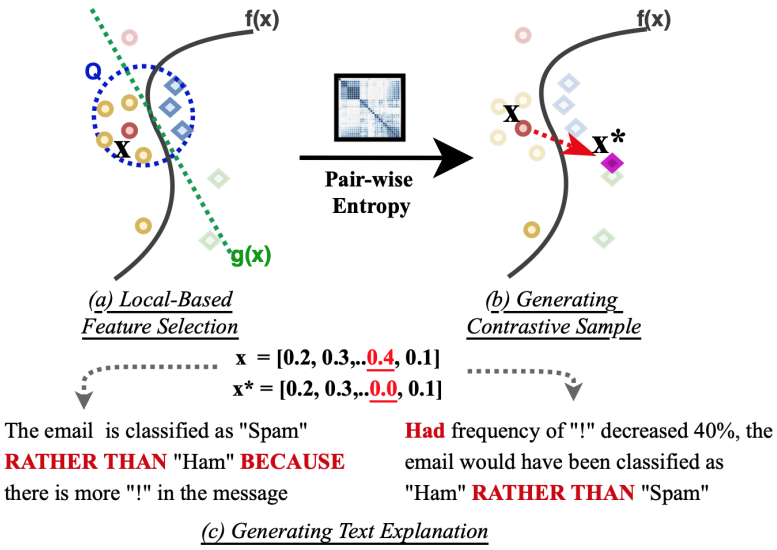


Figure 3.1: GRACE with Local-Based Feature Ranking

Since feature perturbation based on $\mathbf{r}_v$ does not always guarantee that resulted $\tilde{\boldsymbol{x}}_i$ still maintains in the original feature space, to address constraint Eq. (3.6), we project those adjusted features back on to the original domain of $\mathcal{X}$:

$$\tilde{\boldsymbol{x}}_i \longleftarrow P(\tilde{\boldsymbol{x}}_i, dom(\mathcal{X})) \tag{3.10}$$

where P is a projection which ensures that final $\tilde{\mathbf{x}}$ looks more real (e.g., age feature should be a whole number and > 0). The domain space $dom(\mathcal{X})$ can include the maximum, minimum, and data types (e.g., int, float, etc.) of each feature. These can be either calculated from the original training set or manually set by domain experts.

With a fixed k, Eq. (3.10) does not guarantee that $\tilde{\boldsymbol{x}}$ will always cross the decision boundary to class $\boldsymbol{v}$. Hence, we gradually increase $k \longrightarrow K$ until a contrastive sample is successfully generated, i.e., $\text{argmax}(f(\boldsymbol{x})) \neq \text{argmax}(f(\tilde{\boldsymbol{x}}_i))$ or when $k == K$. Alg. 2 illustrates the steps to generate contrastive samples.

One obvious challenge is how to come up with the ordered list $\mathcal{U}^*$ of features to perturb. Next, we will describe this in detail.

3.4.2 Entropy-Based Forward Feature Ranking

As different $\boldsymbol{x}$ might require a different subset of features to perturb, the first challenge is how to prioritize features that are highly vulnerable to the contrastive class $\boldsymbol{v}$. To do this, we rank all features of $\boldsymbol{x}$ according to their predictive power w.r.t prediction $f(\boldsymbol{x})$, resulting in an ordered list $\mathcal{U}$:

$$\mathcal{U} \longleftarrow W(f, \boldsymbol{x}) \tag{3.11}$$

where $W(\cdot)$ is a feature ranking function. The most straight forward way is to rank all features according to their gradients w.r.t the *nearest* contrastive class $\boldsymbol{v}$ that back-propagates through $f(\boldsymbol{x})$, resulting in $W_{\text{GRADIENT}}(f, \boldsymbol{x})$ that returns the ranking of the following set:

$$\{\nabla f_{\boldsymbol{v}}(\boldsymbol{x}^1), \nabla f_{\boldsymbol{v}}(\boldsymbol{x}^2), ..\nabla f_{\boldsymbol{v}}(\boldsymbol{x}^M)\} \tag{3.12}$$

While this method is straightforward, these gradients capture a global view of feature rankings, rather than being customized to a local vicinity of decision boundary around $\boldsymbol{x}$. To overcome this limitation, we introduce $W_{\text{LOCAL}}(f, \boldsymbol{x})$ to return the ranking of the following set:

$$\{w^1_{g(\boldsymbol{x})}, w^2_{g(\boldsymbol{x})}, ...w^M_{g(\boldsymbol{x})}\} \tag{3.13}$$

with $w^j_{g(\boldsymbol{x})}$ is the feature importance score of the j-th feature returned from an interpretable ML model $g(\boldsymbol{x})$ (e.g., feature weights for logistic regression, Gini-score for decision tree, etc.). $g(\cdot)$ is trained on a subset of data points $\mathcal{Q}$ surrounding $\boldsymbol{x}$ (Figure 3.1a) with *maximum likelihood estimation (MLE)* as the loss function:

$$\min_{\theta_{g(x)}} \frac{1}{|\mathcal{Q}|} \sum_{\boldsymbol{x} \in \mathcal{Q}} f(\boldsymbol{x}) \log(g(\boldsymbol{x})) \tag{3.14}$$

If the prediction of $g(\boldsymbol{x})$ on $\mathcal{Q}$ is very close to that of $f(\boldsymbol{x})$, important features from $g(\boldsymbol{x})$ are more prone to change in $f(\boldsymbol{x})$. $\mathcal{Q}$ can be collected by sampling q nearest data points to $\boldsymbol{x}$ from *each of the predicted classes* by $f(\boldsymbol{x})$ on the training set using *NearestNeighbors(NN)* search algorithm with different distance functions. We set $q = 4$ and use *Euclidean distance* throughout all experiments.

In the aforementioned variants of $W(\cdot)$, each feature is treated independently with each other. However, if a pair of selected features are highly dependent on each other (e.g., frequency of "!" and "!!"), the final generated samples will be less informative. Because of this, also to address constraint Eq.(3.5), we want to generate a new ordered list $\mathcal{U}^*$ as follows:

$$\mathcal{U}^* \longleftarrow W_{Entropy}(\mathcal{X}, \mathcal{U}) \tag{3.15}$$

where $W_{Entropy}$ is a forward-based selection approach, which will *iteratively* add each feature from $\mathcal{U}$ (from the most to least predictive) to $\mathcal{U}^*$ one by one such that the mutual information of *any* pairs of features in $\mathcal{U}^*$ is within a upper-bound γ:

$$SU(\mathcal{X}^i\,,\,\mathcal{X}^j) \leq \gamma \quad \forall i, j \in \mathcal{U}^* \tag{3.16}$$

where $SU \in [0, 1]$ is an entropy-based *Symmetrical Uncertainty* [53] function that measures the mutual information between the i-th and j-th feature of $\mathcal{X}$ as:

$$SU(\mathcal{X}^i\,,\,\mathcal{X}^j) = 2\left[\frac{IG(\mathcal{X}^i \,|\, \mathcal{X}^j)}{H(\mathcal{X}^i) + H(\mathcal{X}^j)}\right] \tag{3.17}$$

where $IG(\mathcal{X}^i \,|\, \mathcal{X}^j)$ is the *information gain* of $\mathcal{X}^i$ given $\mathcal{X}^j$, and $H(\mathcal{X}^i)$, $H(\mathcal{X}^j)$ is empirical entropy of $\mathcal{X}^i$ and $\mathcal{X}^j$, respectively. A value $SU = 1$ indicates that $\mathcal{X}^i$ completely predicts the value of $\mathcal{X}^j$ [54]. In implementation, we apply $SU(\cdot)$ with *normalized* $\mathcal{X}$ to consider the difference in scales across all feature values. Alg. 3 describes function $W_{Entropy}$ in details. One obvious challenge is how to come up with the ordered list $\mathcal{U}^*$ of features to perturb. The next section will describe this in detail.

3.4.3 Generating Explanation Text

After generating contrastive sample $\tilde{\boldsymbol{x}}$, we take a further step and generate an explanation in natural text. Table 3.2 shows generated contrastive samples and the corresponding explanation for various datasets with $K = 5$. To do this, for a specific prediction $f(\mathbf{x})$ and generated contrastive sample $\tilde{\mathbf{x}}$, we first calculate their feature differences, resulting in predicate $\mathcal{P}$ as defined in Def. 1. Then, we can translate $\mathcal{P}$ to text by using condition-based text templates such as ... is classified

Algorithm 3 Feature Selection with Entropy: $W_{Entropy}$

Input: $\mathcal{X}$,$\mathcal{U}$, γ
Output: Ordered list of features $\mathcal{U}^*$

Initialize: $\mathcal{U}^* \longleftarrow \{\}$, $\hat{\mathcal{X}} \longleftarrow$ normalize($\mathcal{X}$)
for i **in** $\mathcal{U}$ **do**
  $to\_add \longleftarrow True$
  for j **in** $\mathcal{U}^*$ **do**
    if $\mathrm{SU}(\hat{\mathcal{X}}^i, \hat{\mathcal{X}}^j) > \gamma$ **then**
      $to\_add \longleftarrow False$
    end if
  end for
  if $to\_add = True$ **then**
    $\mathcal{U}^* \longleftarrow \mathcal{U}^* \cup \{i\}$
  end if
end for
Return $\mathcal{U}^*$

as X RATHER THAN Y because ..., or *h*ad..., it would have been classified as X RATHER THAN Y (Figure 3.1c). Different text templates can be selected randomly to induce diversity in explanation text. The difference in features values can be described in three different degrees of obscurity from (i) *extract value* (e.g., *0*.007 point lower), to (ii) *magnitude comparison* (e.g., *t*wice as frequent), or (iii) *relative comparison* (e.g., *h*igher, lower). Which degree of detail to best use is highly dependent on the specific feature, domain, and the choice of end-users, and they do not need to be consistent among perturbed features in a single explanation text.

Table 3.2: Examples of generated contrastive samples and their explanation texts

Dataset	Features/*Prediction*	Type	Original	Generated	Changes	Explanation Text
cancer95	bare_nuclei *diagnosis*	int	1 benign	10 malignant	↑ 9	"if there were 9 more bare nucleus, the patient would be classified as malignant *RATHER THAN* benign"
spam	word_freq_credit word_freq_money *class*	float float	0.470 0.470 Spam	0.225 0.190 Ham	↓ 0.245 ↓ 0.280	"The message is classified as spam *RATHER THAN* ham because the word 'credit' and 'money' is used twice as frequent as that of ham message"

Table 3.3: Dataset statistics and prediction performance

Size	Dataset	#Class	#Feat.	#Data	Acc.*	F1*
small	eegeye	2	14	14980	0.858	0.858
	diabetes	2	8	768	0.779	0.777
	cancer95	2	9	699	0.963	0.963
	phoneme	2	5	5404	0.774	0.772
	segment	7	19	2310	0.836	0.817
	magic	2	10	19020	0.862	0.859
medium	biodeg	2	41	1055	0.853	0.851
	spam	2	57	4601	0.932	0.932
	cancer92	2	30	569	0.958	0.958
large	mfeat	10	216	2000	0.943	0.936
	musk	2	166	476	0.783	0.789

(*) Accuracy and F1 scores are averaged across 10 different runs.

3.4.4 Complexity Analysis

According to Alg. 1, we analyze the time complexity of GRACE on each prediction instance as follows. The predictive feature ranking step using $\mathcal{W}_{Gradient}$ takes $\mathcal{O}(M \log M)$ with *Quick Sort*. Reordering the ranked list of features with $W_{Entropy}$ takes $\mathcal{O}(M)$. Generating contrastive sample step takes $\mathcal{O}(M) + Z\mathcal{V}$ with $K \ll M$, where Z is total number of classes to predict, and $\mathcal{V}$ is the time complexity of forward and backward pass of $f(\boldsymbol{x})$. Generating an explanation text then takes another $\mathcal{O}(M)$. To sum up, the overall time complexity of GRACE to generate an explanation for each prediction instance is $\mathcal{O}(M \log M) + Z\mathcal{V}$ with $K \ll M$ (excluding the overhead of training $g(\cdot)$ and searching for $\mathcal{Q}$ in case of W_{Local}).

3.5 Experiments

In this section, we conduct experiments to verify the effectiveness of GRACE. Specifically, we want to answer two questions: (i) how good are the generated interventive contrastive samples? and (ii) how good are the generated explanation?

3.5.1 Datasets

We select 11 publicly available datasets of different domains and scales from [55] to fully evaluate and understand how well GRACE works with neural networks trained on data with varied properties (Table.5.3). As shown in Table.5.3, the

datasets are grouped into three scale levels according to the number of features. Each dataset is split into training, validation, and test set with a ratio of 8:1:1, respectively.

3.5.2 Compared Methods

Since our proposed framework combines the best of both worlds: adversarial generation and neural network model explanation, we select various relevant baselines from two aspects.

- NearestCT: Instead of generating synthetic contrastive sample for explanation for data point $\boldsymbol{x}$, this approach selects the *nearest* contrastive samples of $\boldsymbol{x}$ from the *training set* to provide contrastive explanation for the prediction $f(\boldsymbol{x})$.
- DeepFool [51]: An effective approach that was originally proposed for untargeted attack by generating adversarial samples. Even though DeepFool is not designed for generating samples to explain predictions, we consider this as a baseline that intervenes on *all features* to generate contrastive samples.
- Lime [4]: A *local interpretable model-agnostic explanation* approach that provides explanation for individual prediction. This approach replies on visualization of feature importance scores (for text and tabular data), and feature heat-map (for image data) to deliver explanation. We use an out-of-the-box toolkit to run experiments for comparison. Lime is selected mainly due to its popularity as a baseline for ML explanation approach.
- GRACE-Gradient and GRACE-Local (**ours**): GRACE with predictive feature ranking function W as $W_{Gradient}$ and W_{Local}, respectively.

`https://github.com/marcotcr/lime`

Table 3.4: All results are averaged across 10 different runs. The best and second best results are highlighted in **bold** and underline.

Statistics	Dataset	# Features < 30						30 ≤ # Features < 100			100 ≤ # Features	
		eegeye	**diabetes**	**cancer95**	**phoneme**	**segment**	**magic**	**biodeg**	**spam**	**cancer92**	**mfeat**	**musk**
$\mathbf{R}_{\text{avg\#Feats}}$	NearestCT	13.56	6.93	5.92	4.82	16.10	9.97	20.53	17.50	29.97	204.22	147.86
	DeepFool	14.00	8.00	9.00	5.00	19.00	10.00	41.00	57.00	30.00	216.00	166.00
	GRACE-Local	1.15	**1.55**	2.7	**1.25**	**2.42**	1.68	3.07	2.95	**3.95**	3.28	3.74
	GRACE-Gradient	**1.0**	1.96	**2.66**	1.3	3.84	**1.6**	**1.93**	**1.09**	4.5	**2.76**	**2.85**
$\mathbf{R}^{*}_{\text{info}-\text{gain}}$	NearestCT	0.69	0.44	**0.64**	0.12	0.19	0.04	0.44	0.62	0.02	0.58	0.28
	DeepFool	**0.7**	0.41	0.62	0.12	0.33	0.05	0.58	0.53	0.01	**0.59**	0.29
	GRACE-Local	0.64	**0.79**	0.49	**0.81**	**0.55**	0.67	0.46	0.47	**0.13**	0.34	0.3
	GRACE-Gradient	0.64	0.62	0.52	0.78	0.23	**0.71**	**0.76**	**0.95**	0.04	0.50	**0.4**
$\mathbf{R}_{\text{influence}}$	NearestCT	0.05	0.06	0.11	0.03	0.01	0.00	0.02	0.04	0.00	0.00	0.00
	DeepFool	0.05	0.05	0.07	0.02	0.02	0.00	0.01	0.01	0.00	0.00	0.00
	GRACE-Local	0.55	**0.52**	0.18	**0.65**	**0.23**	0.4	0.15	0.16	**0.04**	0.1	0.08
	GRACE-Gradient	**0.64**	0.33	**0.2**	0.61	0.06	**0.45**	**0.4**	**0.88**	0.01	**0.18**	**0.14**

3.5.3 Evaluation of Generated Samples

In this section, we want to examine the quality of generated contrastive samples. Since DeepFool, NearestCT and GRACE generate intermediate samples to explain predictions, while Lime is not, we compare and analyze Lime separately in Section 3.5.4 to evaluate final generated explanation.

For each dataset, we train a neural network model $f(\cdot)$ using the training set. We tune it using the validation set together with early-stopping strategy to prevent overfitting and report its performance on the test set. Table 5.3 reports the averaged performance across 10 different runs. We set $K = 5, \gamma = 0.5$, and generate $\tilde{\boldsymbol{x}}$ to explain predictions $f(\boldsymbol{x})$ of all samples in test set, resulting in the set of generated contrastive samples $\tilde{\mathcal{X}}$.

To thoroughly examine the proposed approach, we come up with the following analytical questions (AQs).

AQ1 **Fidelity**: How accurate are the generated contrastive samples' labels, i.e., whether they can cross neural network model's decision boundary as expected?

AQ2 **Conciseness**: How concise are generated samples, i.e., how many features needed to be perturbed to successfully generate contrastive samples?

AQ3 **Info-gain**: How informative are generated samples?

AQ4 **Influence**: Derived from Def. 2, how well do the generated samples answer the question *Why X rather than Y?*.

3.5.3.1 AQ1. Fidelity

Fidelity, measured by $\mathbf{R}_{\text{fidelity}}$, shows how accurately contrastive samples are generated according to the neural network model's boundary, i.e., the accuracy of generated samples' labels w.r.t their predictions by the neural network model:

$$\mathbf{R}_{\text{fidelity}} = \frac{1}{|\tilde{\mathcal{X}}|} \sum_{(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \in \tilde{\mathcal{X}}} \mathbb{1}(\tilde{\mathbf{y}} = \text{argmax}(f(\tilde{\mathbf{x}}))) \tag{3.18}$$

Different from the two baselines, two variants of GRACE have to satisfy the domain constraints, minimize the number of features, and their entropy, all at the same time. Nevertheless, with $K = 5$, our method shows an average $\mathbf{R}_{\text{fidelity}}$

of around 0.8 for both GRACE-GRADIENT and GRACE-LOCAL. As the # of perturbed features increases, the Fidelity scores for both GRACE-GRADIENT and GRACE-LOCAL also increase, which satisfies the expectation (Figure 3.2).

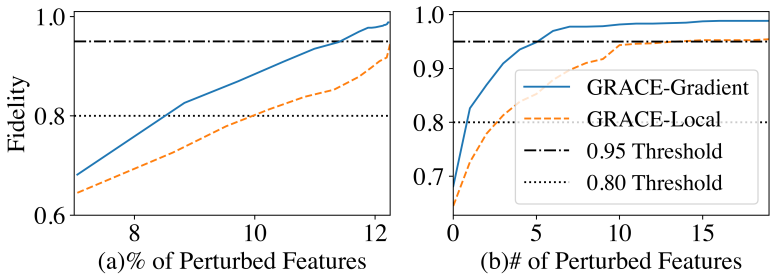


Figure 3.2: Percentage of perturbed features v.s fidelity

3.5.3.2 AQ2. Conciseness

We not only want to generate samples with high *fidelity*, but also want to perturb as few features as possible. Thus, we introduce *conciseness* that measures the ability to generate $\tilde{\boldsymbol{x}}$ by changing as few features as possible. To do this, we want to see how *fidelity* correlates with the average number of perturbed features, denoted as $\mathbf{R}_{\text{avg\#Feats}}$:

$$\mathbf{R}_{\text{avg\#Feats}} = \frac{1}{|\tilde{\mathcal{X}}|} \sum_{\tilde{\mathbf{x}} \in \tilde{\mathcal{X}}} |\mathcal{S}_{\tilde{\boldsymbol{x}}}| \tag{3.19}$$

where $\mathcal{S}_{\tilde{\boldsymbol{x}}}$ returns the list of features to perturb in $\boldsymbol{x}$ to generate $\tilde{\boldsymbol{x}}$.

Table 3.4 shows that our GRACE framework dominates the two baselines DEEPFOOL and NEARESTCT on $\mathbf{R}_{\text{avg\#Feats}}$ by large margin. Specifically, with $K = 5$, our approach is able to generate contrastive samples with much less number of perturbed features, averaging around less than 2.5 features across all datasets. Interestingly, GRACE-GRADIENT was able to change on average less than 3 out of a total of 216 and 166 features in the case of *mfeat* and *musk* dataset, respectively.

Moreover, while our method only needs to use as few as 12.5% of total # of features to achieve fidelity of around 0.95 (Figure 3.2), DEEPFOOL and NEARESTCT baseline needs to change almost 100% of the total # of features to achieve a similar score.

3.5.3.3 AQ3. Info-gain

Since we want to generate samples that are informative, we hope to minimize the averaged mutual information of all pairs of selected features across all samples in $\tilde{\mathcal{X}}$. Hence, we formulate $\mathbf{R}_{\text{info-gain}}$ to measure such characteristic of being informative as follows:

$$\mathbf{R}_{\text{info-gain}} = 1 - \frac{1}{|\tilde{\mathcal{X}}|} \sum_{\tilde{\mathbf{x}} \in \tilde{\mathcal{X}}} \sum_{i \in \mathcal{S}_{\tilde{\mathbf{x}}}} \sum_{j \in \mathcal{S}_{\tilde{\mathbf{x}}}} \frac{\text{SU}\left(\mathcal{X}^i, \mathcal{X}^j\right)}{|\mathcal{S}_{\tilde{\mathbf{x}}}|^2} \tag{3.20}$$

To be fair with other baselines, we instead report $\mathbf{R}^*_{\text{info-gain}} = \mathbf{R}_{\text{info-gain}} \times \mathbf{R}_{\text{fidelity}}$ to take into account the fidelity score. Even so, thanks to the entropy-aware feature selection mechanism, GRACE is able to generate contrastive samples that are much more informative compared to DeepFool's in most of the datasets (Table 3.4). This shows that samples generated by our framework are not only contrastive but also informative to the final explanation.

3.5.3.4 AQ4. Influence

Extended from influence score with tolerance parameter $\lambda = 1$ (Def. 2), we aim to measure how well the generated samples can influence the explanation of a specific prediction. Denoted by $\mathbf{R}_{\text{influence}}$, the influence score first captures whether generated samples are still within the original domain space, or $\mathbf{R}_{\text{domain}}$ as follows:

$$\mathbf{R}_{\text{domain}} = \frac{1}{|\tilde{\mathcal{X}}|} \sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}} \mathbb{1}(\tilde{\boldsymbol{x}} \in \text{dom}(\mathcal{X})) \tag{3.21}$$

Moreover, the influence score is also proportional to how faithful generated samples are to the neural network model's decision boundary ($\mathbf{R}_{\text{fidelity}}$), how informative they are ($\mathbf{R}_{\text{info-gain}}$), how concise in terms of number of perturbed features ($\mathbf{R}_{\text{avg\#Feats}}$), resulting in a $\mathbf{R}_{\text{influence}}$ calculated as follows:

$$\mathbf{R}_{\text{influence}} = \frac{\mathbf{R}_{\text{fidelity}} \times \mathbf{R}_{\text{info-gain}} \times \mathbf{R}_{\text{domain}}}{\mathbf{R}_{\text{avg\#Feats}}} \tag{3.22}$$

Intuitively, $\mathbf{R}_{\text{influence}}$ describes the capability to generate new contrastive samples that are both informative, concise, and valid within the original domain space. Hence, the larger the score, the better.

Regarding $\mathbf{R}_{\text{domain}}$, experiments show that DeepFool performs worst on $\mathbf{R}_{\text{domain}}$,

averaged about 0.86, since the generation might move $\tilde{\mathbf{x}}$ much further away from the original distribution, while other methods always ensure that generated samples are within the original domain space. As regards as $\mathbf{R}_{\text{influence}}$, GRACE is able to generate highly more influential contrastive samples than DeepFool and NearestCT even when taking $\mathbf{R}_{\text{fidelity}}$ into account, which is the strongest point of both two baselines.

3.5.4 Evaluation of Generated Explanation

In this section, we want to compare GRACE with Lime [4] from end-users' perspectives on their generated explanation. Before introducing user-studies to compare between two methods, we first draw some observations in a case-study below.

3.5.4.1 Case-study: breast-cancer diagnosis

We select *cancer95* dataset to experiment. Following the same experimental setting in Section 3.5, we apply Lime and GRACE on the trained neural network model to explain its predictions on the test set. Figure **??** depicts explanation produced by Lime on a patient diagnosed as malignant by the model. Following guideline published by Lime's author , explanation for each feature can be interpreted as follows: ″if bare_nuclei is less than or equal 6.0, on average, this prediction would be 0.15 less malignant". With the same prediction, GRACE generates an explanation text as follows: ″Had bare_nuclei been 7.0 point lower and clump_thickness been 9.0 point lower, the patient would have been diagnosed as **benign rather than malignant**"

Method wise, both are *instance-based* explanation algorithms, or to explain individual predictions. From Figure **??**, by presenting top-k important features, Lime does not convince if and how a single or combinations of features are vulnerable to the contrastive class, but this is very vivid and concise in case of GRACE. Moreover, while both methods provide some intuition on decisive thresholds at which the prediction would change its direction, the thresholds provided by Lime is *only* only a local approximation of the neural network model, while that provided by GRACE (e.g., 7.0 point lower for bare_nuclei) is faithful to the neural network model (*fidelity score* is 1.0). Overall, the explanation generated by GRACE

`https://github.com/marcotcr/lime`

Table 3.5: User-study with hypothesis testing to compare explanation generated by GRACE against LIME

	Alternative Hypothesis	t-stats	p-value	df
$\mathcal{H}_1$	GRACE is more intuitive and friendly	2.3115	0.0104*	42
$\mathcal{H}_2$	GRACE is more comprehensible	3.0176	0.0013**	42
$\mathcal{H}_3$	GRACE leads to more accurate actions	4.4875	$3.39e^{-5}$**	37

**reject Null hypothesis with p-value<0.05 (95% CI) on one-tailed t-test*
***reject Null hypothesis with p-value<0.0 1 (99% CI) on one-tailed t-test*

is more concise and faithful to the decision boundary of the neural network model.

3.5.4.2 User-Study 1: Intuitiveness, friendliness & comprehensibility

: We have recruited participants on Amazon Mechanical Turk (AMT) and asked them to compare two explanation methods: LIME and GRACE. Without assuming or requiring any ML background on the participants, we want to test two alternative hypothesises: explanation generated by GRACE is (H_1) more intuitive and friendly, and (H_2) more comprehensible than that generated by LIME to general users. To test these, using the same prediction instance, we first generate explanation text by LIME and GRACE. While by default LIME returned explanation for top 10 features, we limit only 5 features that are the most significant. On the contrary, GRACE only needs 2 features for generating contrastive explanation. Since LIME method originally does not generate explanation text, we then translate its result to text interpretation as described in previous case-study. Finally, we ask the participants to rank on a scale from 1 to 10 for each question (i) and (ii). We did the surveys for each method *separately*.

From Table 3.5 and Figure 3.4, it is significant ($p\text{-}value \leq 0.05$) that GRACE is able to generate more intuitive, friendly ($\mathcal{H}_1$, 6.35 v.s. 4.76 in mean ranking), and more comprehensible ($\mathcal{H}_2$, 7.35 v.s. 5.52 in mean ranking) explanation than LIME for general users.

3.5.4.3 User-Study 2: How much end-users indeed understand the explanation?

In practice, ML models usually play the role of assisting human to make informative decisions [46]. Therefore, extending from previous experiment, we hypothesize that a good explanation should not only be comprehensible, but should also help

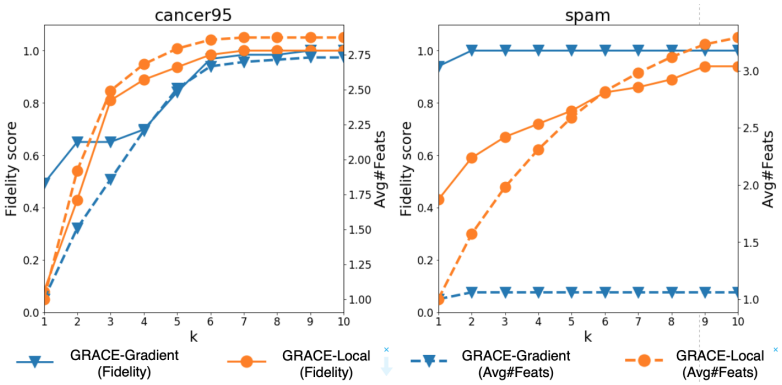


Figure 3.3: Effects of K on $\mathbf{R}_{\text{fidelity}}$ and $\mathbf{R}_{\text{avg\#Feats}}$ score

materialize in accurate decision. Here we want to test workers' actual understanding from the explanation with alternative hypothesis (H_3): users who are provided with explanation generated by GRACE are better at making post-explanation decision than those provided with explanation generated by LIME. To test this, we first present the same prediction scenario from previous user-study, then ask each participant to analyze the explanation and adjust the sample's feature values such that the model would change its prediction (e.g., from malignant to benign). This task requires the worker to recognize from the explanation hints of both (i) what the key features are and (ii) how the changes of those features affect the model's prediction. To ensure the quality of the workers, we only select workers with "US Graduate Degree" as a qualification provided by AMT. We use the trained model to validate the responses and report the average accuracy.

From Table 3.5 and Figure 3.4, it is highly significant ($p\text{-}value \leq 0.01$) that workers who provided with explanation generated by GRACE have more accurate answers than those provided with explanation generated by LIME ($\mathcal{H}_3$), showing 0.75 v.s 0.16 of average accuracy, respectively. In other words, explanation generated by GRACE is more effective in supporting users to make tangible decisions, such as suggesting an alternative scenarios when dealing with neural network models.

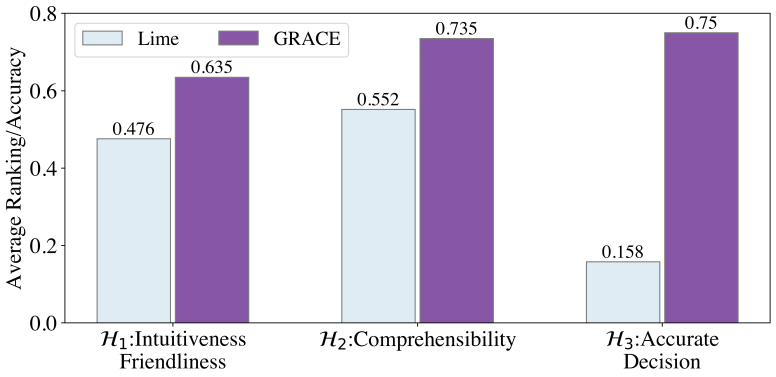


Figure 3.4: Comparison of generated explanation: GRACE v.s. LIME. Scores are normalized to [0,1]

3.5.5 Parameter Sensitivity Analysis

3.5.5.1 Effects of K

One of the important factors that largely affect the explainability of GRACE is the value of parameter K, or the **maximum** number of features to change during the contrastive samples generation process.

While a small K is more preferable, it would become more challenging for GRACE to ensure perturbed samples to cross the decision boundary. This will eventually hurt $\mathbf{R}_{\text{fidelity}}$. Here we want to see how different K values affect the generated samples' *fidelity* and the number of perturbed features. For each dataset, we next train a neural network model and test this model with all values of K = *{1, 2, 3,.. 10}* and plot it against respective $\mathbf{R}_{\text{fidelity}}$ and $\mathbf{R}_{\text{avg\#Feats}}$.

Figure 3.3 reports two distinctive patterns between GRACE-GRADIENT and GRACE-LOCAL: (i) both approaches witness gradual increment in $\mathbf{R}_{\text{fidelity}}$ and $\mathbf{R}_{\text{avg\#Feats}}$, with neither one of them dominates the performance (e.g., *cancer95* dataset), or (ii) one of them greatly out-weights the other (e.g., *spam* dataset). Overall, by increasing K, generated samples are more faithful to the neural network model' decision boundary, yet the average number of features needed to change to achieve so also increases, hence eventually reduce explainability.

Table 3.6: Effects of entropy threshold γ on $\mathbf{R}_{\text{info}-\text{gain}}$

Dataset	Method	1.0	0.7	0.5	0.3
musk	GRACE-GRADIENT	0.51	0.51	**0.58**	**0.58**
	GRACE-LOCAL	0.36	0.36	**0.54**	**0.54**
segment	GRACE-GRADIENT	0.57	0.57	**0.59**	**0.59**
	GRACE-LOCAL	0.79	0.79	**0.84**	**0.84**

3.5.5.2 Effects of entropy threshold γ

Entropy threshold γ is set to ensure that no pairs out of selected features are conveying very similar information, hence making generated samples more informative to users. Similar to the previous experiment, we keep other parameters the same while vary γ as $\{1.0, 0.7, 0.5, 0.3\}$. The results are shown in Table 3.6. We observed that γ is not very sensitive, showing the best value of $\gamma \leq 0.5$, which can be explained that the pair of features are usually more or less predictable given the other at a specific level. However, by setting $\gamma = 0.5$, we can observe larger improvement in case of *musk* and *segment* dataset.

3.6 Related Work

Regarding *explanation by intervention*, our Def. 1 relates to *Quantitative Input Influence* [56], a general framework to quantify the influence of a set of inputs on the prediction outcomes. The framework follows a two-step approach: (i) it first changes each individual feature by replacing it with a random value, and *then* (ii) observes how the outcome, i.e., prediction, changes accordingly. However, we propose a more systematic way by generating a new sample at once by directly conditioning it on a contrastive outcome (X *rather than* Y). A few prior works (e.g., [52, 57, 58]) also propose to generate contrastive samples with (i) minimal corrections from its original input by minimizing the distance: $\delta = \|\boldsymbol{x} - \tilde{\mathbf{x}}\|_p$ and with (ii) minimal number of features needed to change to achieve such corrections. While Wachter et al. [57] use δ with ℓ_1 norm to induce sparsity with the hope to achieve (ii), Zhang et al. [58] approach the problem in a reverse fashion, in which they try to search for minimal δ w.r.t to a pre-defined number of features to be changed. Regardless, without considering the mutual information among pair-wise of features, it does not always guarantee that generated samples are informative to

end-users. The work [59] also proposes a method to use decision trees to search for a decisive threshold of feature's values at which the prediction will change, and utilize such threshold to generate explanations for neural network model' predictions. While sounds similar to our approach, this method shares a similar dis-merit with LIME [4] since the generated explanation is only an approximation and not faithful to the model. In this chapter, we take a novel approach to generate contrastive samples that are not only contrastive but also faithful to the neural network model and "informative" to end-users.

As regards as *features selection*, we employ a forward-based approach together with *Symmetrical Uncertainty (SU)* and the approximation of features importance according to the neural network model. While there are other algorithms for ranking or selecting features (e.g., submodularity [60], ℓ_1 [57], tree-based ([59], etc.), our proposed method is selected because of it is both effective (high fidelity and informative scores as a result) and easy to implement, not to mention that SU can work with continuous features, and it also considers the bias effects in which one feature might have many diverse values than the other [54].

3.7 Limitation and Future Work

There are several Interesting future directions. First, in this chapter, we intervene a selected subset of features without considering conditional dependencies among all variables after such intervention. This might create undesirable samples that are unrealistic (e.g., "'a pregnant man"). Thus, we plan to address interactions among the features to generate samples that are *more realistic*. Second, in this work, we assume a white-box setting that we can access the gradients of the model. We want to extend GRACE for other black-box settings, gradients of which are not accessible. Since our method works exclusively for multinomial classification task, we also plan to apply it on other ML tasks such as regression, clustering, etc.

3.8 Conclusion

In this chapter, we borrow "contrastive explanation" and "explanation by intervention" concepts from previous literature and develop a generative-based approach to explain neural network models' predictions. We introduce GRACE, a novel

instance-based algorithm that provides end-users with simple natural text explaining neural network models' predictions in a contrastive *"Why X rather than Y"* fashion. To facilitate such an explanation, GRACE extends adversarial perturbation literature with various conditions and constraints, and generates contrastive samples that are concise, informative and faithful to the neural network model's specific prediction. User-studies and quantitative experiments on several datasets of varied scales and domains have demonstrated the effectiveness of the proposed approach.

Chapter 4
Learning to Attack: Generating Malicious Comments to Attack Neural Fake News Detection Models

4.1 Background

Circulation of fake news, i.e., false or misleading pieces of information, on social media is not only detrimental to individuals' knowledge but is also creating an erosion of trust in society. Fake news has been promoted with deliberate intention to widen political divides, to undermine citizens' confidence in public figures, and even to create confusion and doubts among communities [61]. Hence, any quantity of fake news is intolerable and should be carefully examined and combated [62]. Due to the high-stakes of fake news detection in practice, therefore, tremendous efforts have been taken to develop fake news detection models that can auto-detect fake news with high accuracies [63–66]. Figure **??** (on top) shows an example of a typical news article posted on the social media channels such as Twitter and Facebook. A fake news detection model then uses different features of the article (e.g., headline and news content) and outputs a prediction on whether such an article is real or fake. Further, recent research has shown that users' engagement (e.g., user comments or replies) on public news channels on which these articles are shared become a critical signal to flag questionable news [64]. Hence, some

of the state-of-the-art (SOTA) fake news detection models [64–67] have exploited these user engagement features into their prediction models with great successes.

Despite the good performances, the majority of SOTA detectors are deep learning based, and thus become vulnerable to the recent advancement in adversarial attacks [68]. As suggested by [69], for instance, a careful manipulation of the title or content of a news article can mislead the SOTA detectors to predict fake news as real news and vice versa. [70] also shows that hiding questionable content in an article or replacing the source of fake news to that of real news can also achieve the same effect. However, these existing attack methods suffer from three key limitations: (i) unless an attacker is also the publisher of fake news, she cannot exercise *post-publish* attacks, i.e., once an article is published, the attacker cannot change its title or content; (ii) an attacker generates adversarial texts either by marginally tampering certain words or characters using pre-defined templates (e.g., "hello" → "he11o", "fake" → "f@ke" [71]), appending short random phrases (e.g., "zoning tapping fiennes") to the original text [72], or flipping a vulnerable character or word (e.g. "opposition" → "oBposition") [73], all of which can be easily detected by a careful examination with naked eyes; and (iii) they largely focus on the vulnerabilities found in the title and content, leaving social responses, i.e., *comments* and *replies*, unexplored.

Since many SOTA neural fake news detectors exploit users' comments to improve fake news detection, this makes them highly vulnerable from attacks via adversarial comments. Figure **??** shows an example of such an attack. Before the attack, a fake news detector correctly identifies a real article as real. However, using a malicious comment as part of its inputs, the same detector is misled to predict the article as fake instead. Compared with manipulating news title or content, an attack by adversarial comments have several advantages: (i) *accessibility:* as it does not require an ownership over the target article, an attacker can easily create a fake user profile and post malicious comments on any social media news posts; (ii) *vulnerability:* it is less vulnerable than attacking via an article's title or content, as the comments written by general users often have a higher tolerance in their writing quality (e.g., using more informal language, slang, or abbreviations is acceptable in user comments) compared to that of an article's title or content. This makes any efforts to detect adversarial comments more challenging. Despite these advantages, to our best knowledge, there exist few studies on the vulnerability of

Figure 4.1: A malicious comment generated by MALCOM misleads a neural fake news detector to predict real news as fake.

neural fake news detectors via malicious comments.

Therefore, in this chapter, I formulate a novel problem of adversarial comment generation to fool fake news detectors. Generating adversarial comments is non-trivial because adversarial comments that are misspelled or irrelevant to the news can raise a red flag by a defense system and be filtered out before it has a chance to fool fake news detector. Thus, we are faced with two challenges: (i) *how to generate adversarial comments that can fool various cutting-edge fake news detectors to predict target class?*; and (ii) *how to simultaneously generate adversarial comments that are realistic and relevant to the article's content;* In an attempt to solve these challenges, we propose **MALCOM**, a novel framework that can generate realistic and relevant comments in an end-to-end fashion to attack fake news detection models, that works for both black box and white box attacks. The main contributions are:

- This is the first work proposing an attack model against neural fake news detectors, in which adversaries can post malicious comments toward news articles

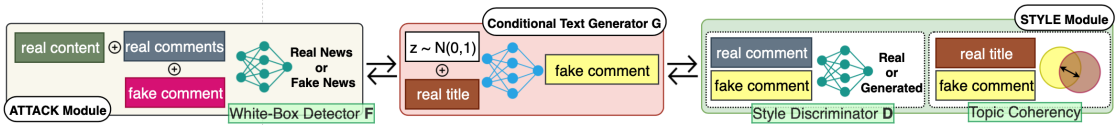


Figure 4.2: MALCOM Architecture.

to mislead cutting-edge fake news detectors.

- Different from prior adversarial literature, our work generates adversarial texts (e.g., comments, replies) with *high quality* and *relevancy* at the sentence level in an end-to-end fashion (instead of the manipulation at the character or word level).

- Our model can fool five top-notch neural fake news detectors to *always* output real news and fake news 94% and 93.5% of the time on average. Moreover, our model can mislead black-box classifiers to always output real news 90% of the time on average.

4.2 Problem Formulation

We propose to attack fake news detectors with three phrases. Phrase I: identifying target articles to attack. Phrase II: generating malicious comments. Phrase III: appending generated comments on the target articles. In this chapter, I focus on the *phrase II* of the attack, which is formally defined as follows. Let $f(\cdot)$ be a target neural network fake news classifier. Denote $\mathcal{X} = \{\boldsymbol{x}_i, C_i\}_{i=1}^N$, $\mathcal{Y} = \{y_i\}_{i=1}^N$ as the features of articles and their ground-truth labels (e.g., fake news or real news) of a dataset $\mathcal{D}$ on which $f(\cdot)$ is trained, with N being the total number of articles. Let $\boldsymbol{x}_i^{title}$, $\boldsymbol{x}_i^{content}$, C_i be the title, content, and a list of all comments of $\boldsymbol{x}_i$, respectively. Then, we want to train a generator G such that, given an unseen article $\{\boldsymbol{x}, C\} \notin \mathcal{X}$ and a target prediction label L^*, G generates a set of M malicious comments C^{adv} to achieve the following objectives:

Objective 1: High quality in writing and relevancy: C^{adv} needs to mimic real comments both in writing quality and relevancy to $\boldsymbol{x}$'s content. This will prevent them from being detected by a robust adversarial text defense system (e.g.,

[74, 75]). Even though generating realistic comments [76] is not the main goal of our paper, it is a necessary condition for successful attacks in practice.

Objective 2: Successful attacks: This is the main objective of the attacker. The attacker contaminates a set of an article's existing comments C by **appending** C^{adv} such that $f : \boldsymbol{x}, C^* \mapsto L^*$, where $C^* \longleftarrow C \oplus C^{adv}$ with $\oplus$ denoting concatenating, and L^* is the target prediction label. When $L^* \leftarrow 0$, C^{adv} ensures that, after posted, an article $\boldsymbol{x}$ will not be detected by f as fake (and not to be removed from the news channels). When $L^* \leftarrow 1$, C^{adv} helps demote real news as fake news (and be removed from the news channels). There are two types of attacks: (i) white box and (ii) black box attack. In a white box attack, we assume that the attacker has access to the parameters of f. In a black box attack, on the other hand, the f's architecture and parameters are unknown to adversaries. This leads to the next objective below.

Objective 3: Transferability: C^{adv} needs to be transferable across different fake news detectors. In a black box setting, the attacker uses a surrogate white box fake news classifier f^* to generate C^{adv} and transfer C^{adv} to attack other black box models f. Since fake news detectors are high-stack models, we impose a stricter assumption compared to previous literature (e.g., [71]) where public APIs to target fake news classifiers are inaccessible. In practice, the training dataset of an unseen black box model will be different from that of a white box model, yet they can be highly overlapped. Since fake news with reliable labels are scarce and usually encouraged to be publicized to educate the general public (e.g., via fact-check sites), fake news defenders have incentives to include those in the training dataset to improve the performance of their detection models. To simplify this, we assume that both white box and black box models share the same training dataset.

4.3 Adversarial Comments Generation

In this chapter, I propose MALCOM, an *end-to-end* ***Mal****icious* ***Com****ment Generation Framework*, to attack fake news detection models. Figure 5.1 depicts the MALCOM framework. Given an article, MALCOM generates a set of malicious comments using a conditional text generator G. We train G together with STYLE

and ATTACK modules. While the STYLE module gradually improves the writing styles and relevancy of the generated comments, the ATTACK module ensures to fool the target classifier.

4.3.1 Conditional Comment Generator: *G*

$G(\boldsymbol{x}, \mathbf{z})$ is a conditional sequential text generation model that generates malicious comment c^* by sampling one token at a time, conditioned on (i) previously generated words, (ii) article $\boldsymbol{x}$, and (iii) a random latent variable $\mathbf{z}$. Each token is sequentially sampled according to conditional probability function:

$$p(c^*|\boldsymbol{x}; \theta_G) = \prod_{t=1}^{T} p(c_t^*|c_{t-1}^*, c_{t-2}^*, \ldots, c_1^*; \boldsymbol{x}; \mathbf{z}) \tag{4.1}$$

where c_t^* is a token sampled at time-step t, T is the maximum generated sequence length, and θ_G is the parameters of G to be learned. G can also be considered as a conditional language model, and can be trained using *MLE* with the *teacher-forcing* [77] by maximizing the negative log-likelihood (*NLL*) for all comments conditioned on the respective articles in $\mathcal{X}$. We want to optimize the objective function:

$$\min_{\theta_G} \mathcal{L}_G^{MLE} = -\sum_{i=1}^{N} c_i \log p(c_i^*|\boldsymbol{x}_i; \theta_G) \tag{4.2}$$

4.3.2 Style Module

Both writing style and topic coherency are crucial for a successful attack. Due to its high-stake, a fake news detector can be self-guarded by a robust system where misspelled comments or ones that are off-topic from the article's content can be flagged and deleted. To overcome this, we introduce the STYLE module to fine-tune G such that it generates comments with (i) high quality in writing and (ii) high coherency to an article's content.

First, we utilize the *GAN* [78] and employ a comment style discriminator D to co-train with G in an adversarial training schema. We use *Relativistic GAN (RSGAN)* [79] instead of standard GAN loss [78]. In RSGAN, the generator G aims to generate realistic comments to fool a discriminator D, while the discriminator D aims to discriminate whether the comment c is more realistic than randomly

sampled fake data generated by G. Specifically, we alternately optimize D and G with the following two objective functions:

$$\begin{aligned}\min_{\theta_G} \mathcal{L}_G^D &= -\mathbb{E}_{(x,c)\sim p_D(\mathcal{X});z\sim p_z}[log(\sigma(D(c) - D(G(\boldsymbol{x}, z)))] \\ \min_{\theta_D} \mathcal{L}_D &= -\mathbb{E}_{(x,c)\sim p_D(\mathcal{X});z\sim p_z}[log(\sigma(D(G(\boldsymbol{x}, z)) - D(c)))]\end{aligned} \tag{4.3}$$

where σ is a *sigmoid* function, θ_G is the parameters of G and θ_D is the parameters of D. By using D, we want to generate comments that are free from misspellings while resembling realistic commenting styles.

Second, to enhance the relevancy between the generated comments and the article, we minimize the mutual information gap between comments generated by G and the article's titles. Specifically, we use *maximum mean discrepancy (MMD)*, which has been shown to be effective in enforcing mutual information. The loss function can be written as:

$$\begin{aligned}\min_{\theta_G} \mathcal{L}_G^H &= MMD(\mathcal{X}^{title}, G(\mathcal{X})) \\ &= \Big[\mathbb{E}_{\boldsymbol{x},\boldsymbol{x'}\sim p_{\mathcal{D}}(\mathcal{X})}\mathbf{k}(\boldsymbol{x}^{title}, \boldsymbol{x'}^{title}) \\ &+ \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}(\mathcal{X});c^*,c^{*'}\sim G(\boldsymbol{x},z)}\mathbf{k}(c^*, c^{*'}) \\ &- 2\mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}(\mathcal{X}),c^*\sim G(\boldsymbol{x},z)}\mathbf{k}(\boldsymbol{x}^{title}, c^*)\Big]^{\frac{1}{2}}\end{aligned} \tag{4.4}$$

where the MMD compares the distribution $\mathcal{X}^{title}$ of real articles' titles and that of generated comments $G(\mathcal{X})$ by projecting them into Hilbert spaces (RKHS) using a Gaussian kernel $\mathbf{k}$. Intuitively, we want to minimize the information gap between the real titles and the generated comments. Moreover, we use $\boldsymbol{x}^{title}$ (i.e., the title of $\boldsymbol{x}$) instead of $\boldsymbol{x}^{content}$ (i.e., the content of $\boldsymbol{x}$) because: (i) an article's content is usually much longer, hence requiring more computation, (ii) an article's title is a summary of the article's content, and (iii) prior studies (e.g., [80]) show that social media users actually rely more on the headlines rather than the actual content for commenting, sharing, and liking.

4.3.3 Attack Module

This module guides G to generate comments that can fool the target classifier. In a white box setting, the fake news classifier can be directly used to guide the

Algorithm 4 Generating Adversarial Comments Algorithm

1: Pre-train G with *teacher-forcing* and MLE using Eq. (4.2) with *train* set.
2: Pre-train a surrogate fake news classifier f using Eq. (4.3.3) with *train* set.
3: **repeat**
4: Training G with D using Eq. (4.3) in mini-batch from *train* set.
5: Training G using Eq. (4.4) in mini-batch from *train* set.
6: Training G with f using Eq. (4.5) in mini-batch from *train* set.
7: **until** convergence

learning of G. In a black box setting, a surrogate fake news classifier can be used to generate and transfer malicious comments to unseen fake news detectors. We denote $f(\boldsymbol{x}_i, \mathbf{C}_i)$ parameterized by θ_f as the surrogate white box classifier, predicting whether or not $\boldsymbol{x}_i$ is fake news. f can be trained using *binary-cross-entropy loss* over $\mathcal{D}$ as:

$$\min_{\theta_f} \mathcal{L}_f = -\frac{1}{N}\sum_{i}^{N}(\mathbf{y}_i log(f(\boldsymbol{x}_i, C_i))) + (1-\mathbf{y}_i) log(1-f(\boldsymbol{x}_i, C_i))$$

To use this trained model f to guide G, we use signals back-propagated from f to force G to generate a new comment c^* such that $f(\boldsymbol{x}_i, C_i^*)$ (where $C_i^* \longleftarrow C_i \oplus \{c^*\}$) outputs a target prediction label $\mathbf{L}^* \in \{0, 1\}$ for the article $\boldsymbol{x}_i$. Specifically, we want to optimize the objective function:

$$\min_{\theta_G} \mathcal{L}_G^{f(\mathbf{L}^*)} = -\frac{1}{N}\sum_{i}^{N}(\mathbf{L}^* log(f(\boldsymbol{x}_i, C_i^*))) + (1-\mathbf{L}^*) log(1-f(\boldsymbol{x}_i, C_i^*)) \tag{4.5}$$

One obvious attack scenario is for an attacker to promote fake news, i.e., to generate comments to mislead the target classifier to classify fake news as real news ($\mathbf{L}^* \longleftarrow 0$). Adversely, an attacker might also want to fool the target classifier to classify real news as fake news ($\mathbf{L}^* \longleftarrow 1$).

4.3.4 Objective Function of MALCOM

At the end, an attacker aims to generate realistic and relevant comments to attack a target fake news classifier by optimizing objective functions as follows.

$$\min_{\theta_f} \mathcal{L}_f; \quad \min_{\theta_D} \mathcal{L}_D; \quad \min_{\theta_G} (\mathcal{L}_G^{MLE} + \mathcal{L}_G^D + \mathcal{L}_G^H + \mathcal{L}_G^{f(\mathbf{L}^*)}) \tag{4.6}$$

where each term in Eq. (4.6) equally contributes to the final loss function. We use Adam [81] to optimize the objective functions with a mini-batch training approach. Alg. 4 shows the overall training algorithm.

4.3.5 Implementation Details

Training with Discrete Data: We need to back-propagate the gradients of the loss in Eq. (4.3, 4.4, 4.5) through discrete tokens sampled by G from the multinomial distribution c_t^* at each time-step t. Since this sampling process is not differentiable, we employ *Gumbell-Softmax* [82] relaxation trick with a τ parameter (i.e., generation temperature) to overcome this. We refer interested readers on the elaborate discussion of the Gumbell-Softmax technique and the effects of τ on generation quality and diversity to [82, 83].

Generation Strategy: For each article $\boldsymbol{x}$, we generate new comment $c \longleftarrow G(\boldsymbol{x}, z)$ where $z \sim \mathcal{N}(0, 1)$. To minimize the risk of being detected by a defender, an attacker desires to select the best set of comments to attack, especially those that are highly relevant to the target article. Hence, for each article $\boldsymbol{x}$, we sample different z to generate different malicious comments and select c that is the most coherent to the article. To measure such the coherency, we derive function $T_k(c, \boldsymbol{x}^{title})$ which will be introduced in Sec. (4.4.1.5).

Architectures and Parameters Setting: We employ *Relational Memory Recurrent Network (LMRN)* [83,84] and multi discriminative representations (MDR) [83] as the corner-stone of G and D architecture. We also observe that D with a CNN-based architecture works very well in our case. The LMRN model is adapted from the SONNET model. The MDR implementation is publicly available .

4.4 Experiments

In this section, we evaluate the effectiveness of MALCOM and try to answer the following analytical questions (AQs):

AQ1 **Quality, Diversity, and Coherency:** How realistic are the generated comments in terms of their writing styles and as well as coherency to the original

`https://github.com/deepmind/sonnet`
`https://github.com/williamSYSU/TextGAN-PyTorch`

Table 4.1: Dataset Statistics and Details of Target Classifiers and Their Fake News Detection Performance

Dataset	#articles	#comments	#fake	#real
GossipCop	4,792	116,308	1,894	2,898
Pheme	5,476	52,612	1,980	3,486

Classifier	GossipCop		Pheme	
	Accuracy	F1	Accuracy	F1
f_{CNN}	0.74	0.74	0.77	0.77
f_{RNN}	0.70	0.69	0.71	0.71
CSI\t [64]	0.65	0.70	0.61	0.61
textCNN [**?**]	0.68	0.68	0.76	0.76
dEFEND [65]	0.76	0.76	0.78	0.78

articles' contents?

AQ2 **Attack Performance:** How effective are generated comments in attacking white box and black box detectors?

AQ3 **Attack Robust Fake News Detectors:** How effective are generated comments in attacking fake news detectors safe-guarded by a robust comments filtering feature?

AQ4 **Robustness:** How many malicious comments do we need and how early can they effectively attack the detectors?

I plan to release all datasets, codes, and parameters used in our experiments.

4.4.1 Set-Up

4.4.1.1 Datasets

We experiment with two popular public benchmark datasets, i.e., **GossipCop** [85] and **Pheme** [**?**]. GossipCop is a dataset of fake and real news collected from a fact-checking website, *GossipCop*, whereas Pheme is a dataset of rumors and non-rumors relating to nine different breaking events. These datasets are selected because they include both veracity label and relevant social media discourse content on Twitter.

We exclude another popular dataset, PolitiFact, also from [85] because it is much smaller and less diverse in terms of topics

4.4.1.2 Data Processing and Partitioning

For each dataset, we first clean all of the comments (e.g., remove mentions, hashtags, URLs, etc.). We also remove non-English comments, and we only select comments that have length from 5 and 20. We split the original dataset into *train* and *test* set with a split ratio of 9:1. Since PHEME dataset does not include articles' contents, we use their titles as alternatives. Table 4.1 shows the statistics of the post-processed datasets. We use the *train* set to train both G and target fake news classifiers f. All the experiments are done *only* on the *test* set, i.e., we evaluate quality and attack performance of generated comments on *unseen* articles and their ground-truth comments.

4.4.1.3 Target Classifier

.

We experiment MALCOM with SOTA and representative fake news classifiers, which are summarized in Table 4.1. Note that both datasets are challenging ones as SOTA methods can only achieve 0.76 and 0.78 in F1 using the first 10 comments of each article. These classifiers are selected because they cover a variety of neural architectures to learn representations of each article and its comments, which is eventually input to a *softmax* layer for prediction. In all of the following classifiers, we encode the article's content into a feature vector of $\mathbb{R}^{512}$ by using Universal Sentence Encoder (USE) [72], followed by a *fully-connected-network (FCN)* layer.

- $\boldsymbol{f_{CNN}}$: This classifier uses CNN layers to encode each comment into a vector. Then, it concatenates the average of all encoded comments with the feature vector of the article's content as the article's final representation.

- $\boldsymbol{f_{RNN}}$: This classifier uses a RNN layer to model the sequential dependency among its comments, output of which is then concatenated with the vectorized form of the article's content as the article's final representation. We utilize *Gated Recurrent Unit (GRU)* as the RNN architecture because it has been widely adopted in previous fake news detection literatures (e.g., [64, 65]).

- **textCNN** [**?**]: TEXTCNN uses a CNN architecture to encode the mean of vector representations of all of its comments. The output vector is then concatenated with the article's content vector as the article's final representation.

- **CSI\t** [64]: CSI uses *GRU* to model the sequential dependency of textual features of user comments and the network features among users participating in an article's discourse to detect fake news. Different from f_{RNN}, this model does not use an article's content as an input. We use a modified version, denoted as CSI\t, that does not use the network features as such information is not available in both datasets.

- **dEFEND** [65]: Given an article, this algorithm utilizes a co-attention layer between an article's content and comments as input to make a final prediction.

Other methods are surveyed but not included in the experiments because: (i) overall, their accuracies were reported inferior to those of **dEFEND**, **CSI**, (ii) SAME [66] only uses extracted sentiments of the comments, not the whole text as input, (iii) FAKEDETECTOR [86] mainly uses graph-based features, which is not within our scope of study, and (iv) TCNN-URG [67] focuses only on early fake news detection.

4.4.1.4 Compared Attack Methods

We compared MALCOM with representative and SOTA adversarial text generators (Table 4.2).

- **Copycat Attack**: We created this method as a trivial attack baseline. CopyCat randomly retrieves a comment from a relevant article in the train set which has the target label. We use *USE* to facilitate semantic comparison among articles' contents.

- **HotFlip Attack [73]**: This attack finds the most critical word in a sentence and replaces it with a similar one to fool the target classifier. Since HotFlip does not generate a whole sentence but make modifications on an existing one, we first use the comment retrieved by CopyCat as the initial malicious comment.

- **Universal Trigger (UniTrigger) Attack [72]**: It searches and appends a *fixed* and *universal* phrase to the end of an existing sentence to fool a text classifier. In this case, we want to find an universal topic-dependent prefix to prepend to every comment retrieved by CopyCat to attack. For a fair

`https://github.com/Eric-Wallace/universal-triggers`

Table 4.2: Comparison among Attack Methods

Method	**end-to-end generation**	**generalization via learning**	**level of attack**
CopyCat	○	○	sentence
HotFlip	○	○	character/word
UniTrigger	○	●	multi-level
TextBugger	○	○	character/word
MALCOM	●	●	sentence

Table 4.3: Examples of Generated Malicious Comment. Spans in purple and italics are retrieved from the train set and carefully crafted. Spans in blue are generated in end-to-end fashion.

Title	why hollywood won't cast renee zellweger anymore
Content	so exactly what led renée zellweger, an oscar (...)
CopyCat	*her dad gave her a great smile*
+HotFlip	*her dad* ~~***gave***~~ ***got*** *her a great smile*
+UniTrigger	***edit season edit*** *her dad gave her a great smile*
+TextBugger	*her dad* ~~***gave***~~ ***ga ve*** *her a great smile*
MALCOM	**why do we need to ruin this season**

comparison and to ensure the coherency with the target article's content, we restrict replacement candidates to the top q=*30* words (for GossipCop dataset) and q=*30* words (for Pheme dataset) representing the article's topic. q is chosen such that replacement candidates are distinctive enough among different topics. These words and topics are retrieved from a topic modeling function $LDA_k(\cdot)$.

- **TextBugger Attack [71]**: This method generates "bugs", i.e., carefully crafted tokens, to replace words of a sentence to fool text classifiers. This attack also requires an existing comment to attack. Therefore, we first use CopyCat to retrieve an initial text to attack. Next, we search for "bugs" using one of the following strategies *insert, delete, swap, substitute-c, substitute-w* as described in [71] to replace one of the words in a comment that achieves the attack goal.

4.4.1.5 Evaluation Measures

1. **Success Attack Rate (Atk%)**: This quantifies the effectiveness of the attack. For example, a target-real attack on f with Atk% score of 80% indicates that an attacker can fool f to predict *real news* 80% of the time on all news articles that F should have otherwise predicted correctly.

2. **Quality and Diversity**: We use **BLEU** and *negative-log-likelihood loss* **(NLL_gen)** scores to evaluate how well generated comments are in terms of both quality and diversity, both of which are widely adopted by previous text generation literature (e.g., [83,87,88]). While *BLEU* scores depict the quality of the generated text compared with an out-of-sample test set of human-written sentences (the higher the better), *NLL_gen* signals how diverse generated sentences are (the lower the better).

3. **Topic Coherency**: We derive a topic coherency score of a set of arbitrary comments C and its respective set of articles $\mathcal{X}$ of size N as follows: $T_k(X, C) = \frac{1}{N}\sum_{i=0}^{N}[1 - \cos(LDA_k(\boldsymbol{x}_i^{content}), LDA_k(c_i))]$, where $\cos(\cdot)$ is a *cosine similarity* function. $LDA_k(\cdot)$ is a Latent Dirichlet Allocation (LDA) model that returns the distribution of k different topics of a piece of text. We train LDA_k on all of the articles' contents using unsupervised learning with hyper-parameter k. The larger the score is, the more topic-coherent the comment gets to the article. Because different comments generation algorithms work well with different values of k, for a fair comparison, we report the averaged topic coherency across different values of k as the final **Coherency score**: $Coherency = \sum_{k\in\mathcal{K}} T_k(X, C)$. We select $k \in \mathcal{K}$ such that the averaged entropy of topic assignment of each article is minimized, i.e., to ensure that the topic assigned to each article is distinctive enough to have meaningful evaluation.

4.4.2 AQ1. Quality, Diversity and Coherency

Tables 4.3 and 4.4 show the examples of the generated comments by all attacks and their evaluation results on the quality, diversity, and topic coherency. MALCOM generates comments with high quality in writing and topic coherency. However, we do not discount the quality of human-written comments. The reason why BLEU scores of real comments, i.e., CopyCat, are lower than that of MALCOM is because they use a more diverse vocabulary and hence reduce n-gram matching chances with reference text in the *test* set. The user-study in Sec. 4.5.1 will later show that it is also not trivial to differentiate between MALCOM-generated and human-written comments even for human users.

Different from all recent attacks, MALCOM is an end-to-end generation framework, and can control the trade-off between quality and diversity by adjusting the

Table 4.4: Quality, Diversity, Coherency and White Box Attack

Model	**GossipCop Dataset**			
	↑Quality	↓Diversity	↑Coherency	↑Atk%
CopyCat	0.650	-	0.585	0.497
+HotFlip	0.618	-	0.565	0.803
+UniTrigger	0.545	-	0.725	0.929
+TextBugger	0.643	-	0.561	0.749
Malcom\Style	0.740	2.639	0.659	**0.986**
Malcom	**0.759**	**2.520**	**0.730**	0.981
Model	**Pheme Dataset**			
	↑Quality	↓Diversity	↑Coherency	↑Atk
CopyCat	0.697	-	0.578	0.784
+HotFlip	0.657	-	0.530	0.958
+UniTrigger	0.608	-	0.595	0.951
+TextBugger	0.617	-	0.528	0.975
Malcom\Style	0.517	2.399	0.732	**1.000**
Malcom	**0.776**	**1.917**	**0.812**	0.966

"-": NLL_{gen} cannot be computed for retrieval-based method
All experiments are averaged across 3 different runs

τ parameter accordingly (Sec. 4.3.5). Thus, by using a less diverse set of words that are highly relevant to an article, MALCOM can focus on generating comments with both high quality and coherency. The Style module significantly improves the writing style and boosts the relevancy of generated comments. Without the Style module, we observe that a model trained with only the Attack module will quickly trade in writing style for attack performance, and eventually become stuck in mode-collapse, i.e., when the model outputs only a few words repeatedly. We also observe that the Style module helps strike a balance between topic coherency and attack performance.

Table 4.5: Black Box Attack Performance on Different Attack Strategies and Target Classifier Architectures (Atk%)

Attack/Model	**GossipCop Dataset**						**Pheme Dataset**					
	f^*_{RNN}	f_{RNN}	f_{CNN}	CSI\t	textCNN	dEFEND	f^*_{RNN}	f_{RNN}	f_{CNN}	CSI\t	textCNN	dEFEND
Baseline	0.416	0.509	0.499	0.516	0.548	0.652	0.533	0.514	0.498	0.606	0.537	0.543
CopyCat	0.497	0.689	0.688	0.670	0.774	0.802	0.784	0.783	0.766	0.821	0.716	0.644
+HotFlip	0.803	0.813	0.765	0.820	0.838	0.866	0.958	0.850	0.845	0.879	0.811	0.711
+UniTrigger	0.929	0.763	0.722	0.803	0.745	0.817	0.951	0.783	0.782	0.783	0.781	0.730
+TextBugger	0.749	0.736	0.742	0.742	0.784	0.832	0.975	0.832	0.852	0.872	0.823	0.705
Malcom\Style	**0.986**	**0.973**	0.939	0.875	**0.888**	**0.930**	**1.000**	**0.959**	**0.965**	0.880	**0.963**	**0.865**
Malcom	0.981	0.963	**0.941**	**0.911**	0.876	0.912	0.966	0.893	0.893	**0.888**	0.889	0.760

(*) indicates white box attacks. All experiments are averaged across 3 different runs. Malcom\Style: MALCOM *without* the Style module.

4.4.3 AQ2. Attack Performance

In this section, we evaluate Atk% of all attack methods under the most optimistic scenario where the target article is just published and contains no comments. We will evaluate their attack robustness under other scenarios in later sections.

4.4.3.1 White Box Attack

We experiment a white box attack with f_{RNN} target classifier. *RNN* architecture is selected as the white box attack due to its prevalent adoption in various fake news and rumors detection models. Table 4.5 describes white box attack in the first column of each dataset. We can observe that MALCOM is very effective at attacking white box models (98% Atk% and 97% Atk% in GossipCop and Pheme dataset). Especially, MALCOM\Style is able to achieve near perfect Atk% scores. Comparable to MALCOM are UniTrigger and TextBugger. While other attacks such as TextBugger only performs well in one dataset, MALCOM and UniTrigger perform consistently across the two datasets with a very diverse writing styles. This is thanks to the "learning" process that helps them to generate malicious comments from not only a single but a set of training instances. On the contrary, TextBugger for example, only exploits a specific predefined weakness of the target classifier (e.g. the vulnerability where misspellings are usually encoded as unknown tokens [71]) and requires no further learning.

4.4.3.2 Black Box Attack

Let's use a surrogate f_{RNN} model as a proxy target classifier to generate malicious comments to attack black box models. We test their transferability to black box attacks to five unseen fake news classifiers described in Sec. 4.4.1.3. Table 4.5 shows that comments generated by MALCOM does not only perform well on white box but also on black box attacks, achieving the best transferability across all types of black box models. Our method is especially able to attack well-known models such as CSI\t and dEFEND with an average of 91% and 85% of Atk% in GossipCop and Pheme dataset. However, other strong white box attacks such as UniTrigger, TextBugger and HotFlip witness a significant drop in Atk% with black box target classifiers. Particularly, UniTrigger experiences the worst transferability, with its Atk% drops from an average of 94% to merely over 77%

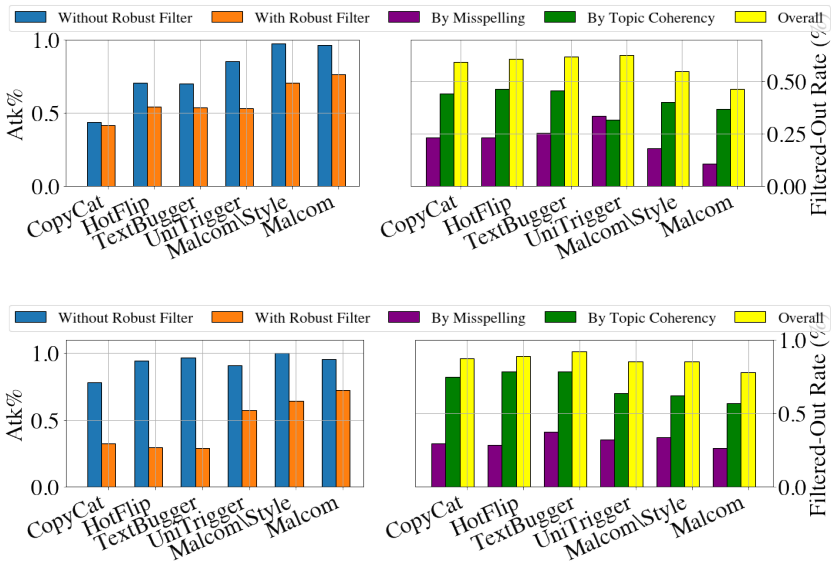


Figure 4.3: Attack Robust Fake News Detector. Top: GossipCop Dataset. Bottom: Pheme Dataset

across all models in both datasets. On the other hand, MALCOM performs as much as 90% Atk% across all black box evaluations. This shows that our method generalizes well not only on fake news detector trained with different set of inputs (e.g., with or without title), but also with different modeling variants (e.g., with or without modeling dependency among comments) and architectures (*RNN*, *CNN*, *Attention*).

4.4.4 AQ3. Attack Robust Fake News Detection

This section evaluates attack performance of all methods under a *post training defense*. This defense comes after the target classifier has already been trained. Before prediction, we use a robust word recognizer called ScRNN [74] to measure the number of misspellings and detect manipulations in the comments. We also use the *Coherency* (Sec. 4.4.1.5) to measure the topic relevancy and set a topic coherency threshold between comments and the target article to filter-out irrelevant comments. We remove any comment that either has more than one suspicious word or have *Coherency* lower than that of the article's title with an allowance margin of 0.05. This defense system is selected because it does not make any

`https://github.com/danishpruthi/Adversarial-Misspellings`

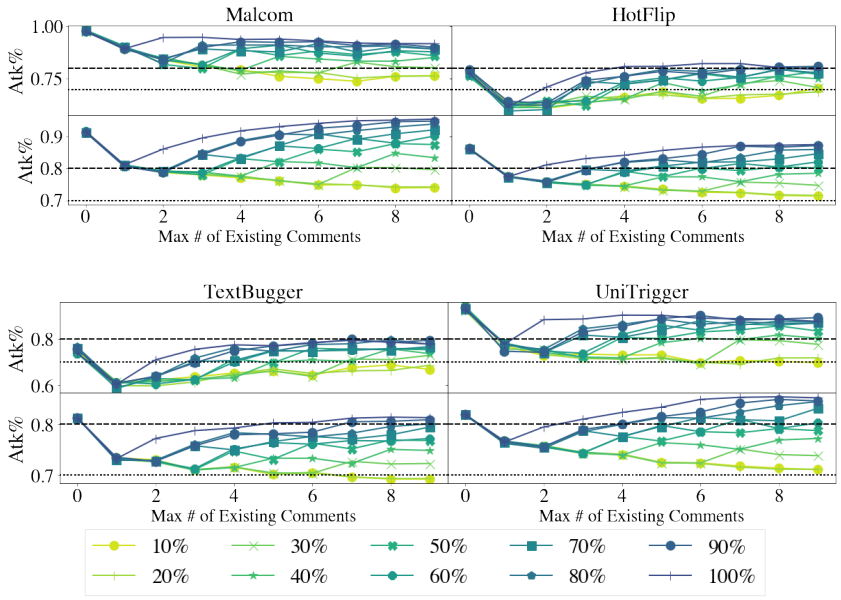


Figure 4.4: Robustness of Intra-Attacks: White Box Setting (First Row) & Black Box (Second-Row) on GossipCop Dataset.

assumption on any specific attack methods, hence it is both general and practical. We measure both Atk% and the *filter-out rate*, i.e., the percentage of comments that are removed by the defense system, for all of the attack methods.

Figure 4.3 shows that our method achieves the best Atk% even under a rigorous defense in both datasets. While Atk% of HotFlip and TextBugger drop significantly (around ↓66% and ↓68%) under the defense, that of MALCOM\Style decreases to only 0.64% from a nearly perfect Atk%. This confirms that the Style module is crucial for generating stealthy comments. Figure 4.3 also shows that MALCOM is the best to bypass the defense system, achieving better, i.e., lower, *filter-out rate* in terms of misspellings compared with real comments retrieved by CopyCat method. This is because MALCOM emphasizes writing quality over diversity to be more stealthy under a robust defense algorithm. Moreover, around 1/3 of real comments selected by CopyCat are filtered-out by ScRNN. This confirms that real comments written on social media are messy and not always free from grammatical errors.

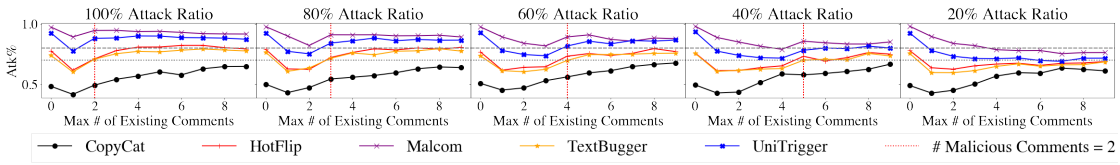


Figure 4.5: Robustness of Inter-Attacks: White Box Setting on GossipCop Dataset

4.4.5 AQ4. Robustness

There is always a trade-off between the # of comments collected for prediction and how early to detect fake news. This section evaluates Atk% w.r.t different # of existing comments for early fake news detection. We evaluate on GossipCop dataset as an example. We assume that the target classifier only has access to a few existing comments, a maximum of 20 in this case. First, we define the *attack ratio* as the ratio of # of malicious comments over # of existing comments. Next, we evaluate the robustness among all attack methods in two categories, namely *Inter- and Intra-Comparison.*

Inter-Comparison: How attack methods perform differently with the same attack ratio? Figure 4.5 shows that MALCOM outperforms other baselines under all attack ratios. Moreover, our method consistently maintains Atk% of at least 80% under a different # of existing comments with an attack ratio as low as 40%. On the contrary, to achieve the same performance, UniTrigger, the second-best attack in terms of robustness, would require a 100% attack ratio.

Intra-Comparison: For each method, how many malicious comments are needed for an effective attack performance under different # of existing comments? Figure 4.4 shows the results on white box and black box attacks. Under the white box attack, all methods display more or less the same performance with more than 1 malicious comment. However, the black box setting observes more variance across different attack ratios. Specifically, MALCOM's performance continuously improves as the # of existing comments increases under any attack ratios $\geq 40\%$. On the contrary, existing attacks show little improvement even with an increasing # of malicious comments.

Table 4.6: Results of User-Study on Generation Quality

Hypothesis	z-score	p-value	Accuracy	#response
$\mathcal{H}_1$	1.4940	0.0676	0.6087	46
$\mathcal{H}_2$	1.2189	0.1114	0.5818	56
$\mathcal{H}_3$	0.9122	0.1808	0.5416	120

4.5 Discussion

4.5.1 Prevent Malicious Comments with Human Support

We examine whether malicious comments generated by MALCOM can be easily flagged by human, i.e., the Turing Test. We use Amazon Mechanical Turk (AMT) to recruit over 100 users to distinguish comments generated by MALCOM (machine-generated) and human. We examine the following alternative hypothesises using one-tailed statistical testing.

1. $\mathcal{H}_1$: Given a comment, the users can correctly detect if the comment is generated by machine (not by human).

2. $\mathcal{H}_2$: Given a comment, the users can correctly detect if the comment is generated by human (not by machine).

3. $\mathcal{H}_3$: Given a machine-generated and a human-written comment, the users can correctly identify the machine-generated.

For quality assurance, we recruit only the users with 95% approval rate, randomly swap the choices and discard responses taking less than 30 seconds. We test on comments generated for 187 unseen and unique articles in the PHEME dataset's *test* set. Table 4.6 shows that we *fail to reject* the null-hypothesises of both $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3$ (p-value > 0.05). While comments generated by MALCOM is not perfectly stealthy (accuracy of $\mathcal{H}_1 > 0.5$), it is still very challenging to distinguish between human-written and MALCOM-generated comments. In fact, human-written comments on social media are usually messy, which lead users to be unable to distinguish between machine-generated ones (accuracy of $\mathcal{H}_2 < 0.6$). Thus, even if human are employed to filter out suspicious or auto-generated comments with a mean accuracy of 60% ($\mathcal{H}_1$), MALCOM can still effectively achieve over

80% Atk% on average with a remaining 40% of the malicious comments (see Sec. 4.4.5). Hence, we need to equip human workers with intensive training to better identify malicious comments. Nevertheless, this can be labor intensive and costly due to a large amount of comments published everyday.

4.5.2 Prevent Malicious Comments with Machine Support

One advantage of the defense system introduced in Sec. 4.4.4 is that filtering out comments based on misspellings and topic coherency does not make any assumption on any specific attack methods, hence it is both general and practical. In the most optimistic scenario where we expects *only* attacks from MALCOM, we can train a ML model to detect MALCOM-generated comments. We use LIWC [89] dictionary to extract 91 psycholinguistics features and use them to train a *Random Forest* classifier to differentiate between human-written, i.e., CopyCat, and MALCOM-generated comments based on their linguistic patterns. The 5-fold cross-validation accuracy is $0.68(+/-0.1)$, which means around 70% and 30% of MALCOM-generated and human-written comments will be flagged and removed by the classifier. From Figure 4.4, if the initial *attack ratio* of 100%, one can then effectively put an upper-bound of around 80% Atk% rate on MALCOM (new *attack ratio* of 40%).

Other ways to defend against MALCOM is to only allow users with verified identifications to publish or engage in discussion threads, or to utilize a fake account detection system (e.g., [90]) to weed out suspicious user accounts. We can also exercise adversarial learning [91] and train a target fake news classifier together with malicious comments generated by potential attack methods. Social platforms should also develop their own proprietary fake news training dataset and rely less on public datasets or fact-checking resources such as *GossipCop* and *PolitiFact*. While this may adversely limit the pool of training instances, it will help raise the bar for potential attacks.

4.5.3 Real News Demotion Attack

An attacker can be paid to either promote fake news or demote real news. By demoting real news, in particular, not only can the attacker cause great distrust among communities, but the attacker can also undermine the credibility of news

Table 4.7: Ablation Test

Models	GossipCop Dataset			
	↑Quality	↓Diversity	↑Coherency	↑Atk%
\Style\Attack	0.645	**1.664**	0.727	0.392
\Attack	0.741	2.032	**0.769**	0.434
\Style	0.740	2.639	0.659	**0.986**
MALCOM	**0.759**	2.520	0.730	0.981
Models	**Pheme Dataset**			
	↑Quality	↓Diversity	↑Coherency	↑Atk%
\Style\Attack	0.759	**1.273**	0.845	0.519
\Attack	0.741	**0.786**	1.431	**0.850**
\Style	0.517	2.399	0.732	**1.000**
MALCOM	0.776	1.917	**0.812**	0.966

\Style, \Attack: MALCOM with the Style, Attack module removed.

organizations and public figures who publish and share the news. In fact, MALCOM can also facilitate such an attack, i.e., to fool fake news detectors to classify real news as fake news, by simply specifying the target label $L^* \longleftarrow 1$ (Sec. 4.3.3). Our experiments show that MALCOM can achieve a real news demotion white box attack with Atk% of around 92% and 95% in GossipCop and Pheme datasets. Figure **??** shows an example of a real news demotion attack. The real news article was posted by *The Guardian*, a reputable news source, on Twitter on June 4, 2018. The article is first correctly predicted as real news by a *RNN*-based fake news classifier. However, the attacker can post a malicious yet realistic-looking comment "*he's a conservative from a few months ago*" to successfully fool the classifier to predict the article as fake instead.

4.5.4 Ablation Test

This section carries out ablation test to show the effectiveness of Style and Attack component of MALCOM. Specifically, we valuate the quality, diversity, coherency and as well as white box attack performance of different variants of MALCOM. Figure 4.7 demonstrates that Style module enhances the writing quality and coherency by large margin from the model without Style module. Especially, Style module is crucial in improving topic coherency score, which then makes generated comments more stealthy under robust fake news defense system (Sec. 4.4.4). Figure 4.7 also shows that Attack module is critical in improving attack performance. While Style and Attack each trades off quality, coherency

with attack success rate and vice versa, the full MALCOM achieves a balanced performance between a good writing style and high attack success. This makes our framework both powerful and practical.

4.5.5 Baselines' Dependency on CopyCat

Compared to MALCOM, one disadvantage of HotFlip, UniTrigger and TextBugger is that they all require an initial comment to manipulate. In theory, manually crafting an initial comment is feasible, yet demands a great labor cost. In practice, an attacker can directly use the target's title or an existing comment as the initial comment to begin the attack. Instead, in this chapter, we use CopyCat to retrieve the initial comment. CopyCat considers both the topic of the target article and the target label into consideration. Hence, it can help complement other baseline attacks in terms of both Atk% and topic coherency. Our experiments show that attacks using comments retrieved by CopyCat achieve much better averaged Atk% across both white box and black box attacks (89% Atk%), compared to the ones using the title (75% Atk%) or a single existing comment (78% Atk%) of the target article. This further justifies the use of CopyCat together with other baseline attacks in our experiments.

4.6 Related Work

4.6.1 Fake News Detection Models

In terms of computation, the majority of works focus on developing machine learning (ML) based solutions to automatically detect fake news. Feature wise, most models use an article's title, news content, its social responses (e.g., user comments or replies) [65], relationships between subjects and publishers [86] or any combinations of them [64]. Specifically, social responses have been widely adopted and proven to be strong predictive features for the accurate detection of fake news [64, 65]. Architecture wise, most detectors use *recurrent neural network (RNN)* [64, 65] or *convolutional neural network (CNN)* [67] to encode either the news content (i.e., article's content or micro-blog posts) or the sequential dependency among social comments and replies. Other complex architecture includes

the use of *co-attention* layers [92] to model the interactions between an article's content and its social comments (e.g., dEFEND [65]) and the adoption of variational auto-encoder to generate synthetic social responses to support early fake news detection (e.g., TCNN-URG [67]).

4.6.2 Attacking Fake News Detectors

Even though there have been several works on general adversarial attacks, very few addressed on the attack and defense of fake news detectors. [69] argues that fake news models purely based on *natural language processing (NLP)* features are vulnerable to attacks caused by small fact distortions in the article's content. Thus, they propose to use a fact-based knowledge graph curated from crowdsourcing to augment a classifier. In a similar effort, [70] examines three possible attacks to fake news detectors. They are hiding questionable content, replacing features of fake news by that of real news, and blocking the classifiers to collect more training samples. The majority of proposed attacks leverage an article's title, content, or source. They assume that the attacker has a full ownership over the fake news publication (thus can change title or content). This, however, is *not* always the case. In this chapter, therefore, I assume a stricter attack scenario where the attacker has no control over the article's source or content, particularly in the case where the attacker is different from the fake news writer. Moreover, we also conjecture that the attacker can be hired to either: (i) promote fake news as real news and (ii) demote real news as fake news to create confusion among the community [61]. To achieve this, instead of focusing on attacking an article's content or source, we propose to generate and inject **new** malicious comments on the article to fool fake news detectors.

4.6.3 Adversarial Text Generation

Text generation is notoriously a complex problem mainly due to the discrete nature of text. Previous literature in text generation include generating clickbaits [2, 76], text with sentiment [93], user responses [67], and fake news [94]. Generating text under adversarial setting, i.e., to attack ML classifiers, is more challenging [88]. Yet there have been tireless efforts to construct adversarial samples to attack text-based ML models [71, 73, 95]. Most of them focus on making marginal modifications (e.g.,

addition, removal, replacement, etc.) in character [71,73] or word level [73,95] of a span of text, either through a set of predefined templates [71] or through a searching mechanism with constraints [73, 95]. Even though these methods have achieved some degree of success, they are only designed for attacking static features such as the title and content of an article. They are not developed for dynamic sequential input like comments where new text can be added over time. Adversarial text generated by previous methods are usually misspelled ("f@ke" v.s. "fake", "lo ve" v.s. "love") [71], or distorted from the original context or meaning (e.g., [71], [73]). Hence, these attacks can easily be filter-out by a robust word recognizer (e.g. [74]) or even by manual visual examination. Because of this, we propose an end-to-end framework to generate stealthy and context-dependent adversarial comments that achieve a high attack performance.

4.7 Limitations and Future Work

In this work, we assume that the attacker and the model provider share the same training dataset. In practice, their training datasets might be overlapped but not exactly the same. Moreover, whether or not comments generated using one sub-domain (e.g., political fake news) can be transferable to another (e.g., health fake news) is also out of scope of this work. Hence, we leave the investigation on the proposed attack's transferability across different datasets for future work. Moreover, we also plan to extend our method to attack graph-based fake news detectors (e.g., [67]), and evaluate our model with other defense mechanisms such as adversarial learning, i.e., to train the target fake news classifier with both real and malicious comments to make it more robust. We also want to exploit similar attack strategy in areas that utilize sequential dependency among text using ML such as fake reviews detection.

4.8 Conclusion

To our best knowledge, this work is the first attempt to attack existing neural fake news detectors via malicious comments. Our method does not require adversaries to have an ownership over the target article, hence becomes a practical attack. We also introduce MALCOM, an end-to-end malicious comments gener-

ation framework that can generate realistic and relevant adversarial comments to fool five of most popular neural fake news detectors to predict fake news as real news with attack success rates of 94% and 90% for a white box and black box settings. Not only achieving significantly better attack performances than other baselines, MALCOM is shown to be more robust even under the condition when a rigorous defense system works against malicious comments. We also show that MALCOM is capable of not only promoting fake news but also demoting real news. Due to the high-stakes of detecting fake news in practice, in future, we hope that this work will attract more attention from the community towards developing fake news detection models that are accurate yet resilient against potential attacks.

Chapter 5
Learning to Defend: Using Honeypots to Proactively Detect Universal Trigger's Adversarial Attacks

5.1 Background

Adversarial examples in NLP refer to carefully crafted texts that can fool predictive machine learning (ML) models. Thus, malicious actors, i.e., attackers, can exploit such adversarial examples to force ML models to output desired predictions. There are several adversarial example generation algorithms, most of which perturb an original text at either character (e.g., [71,96]), word (e.g., [**?**,73,95–97], or sentence level (e.g., [5, 98, 99]).

Because most of the existing attack methods are instance-based search methods, i.e., searching an adversarial example for each specific input, they do not usually involve any learning mechanisms. A few *learning-based* algorithms, such as the Universal Trigger (*UniTrigger*) [95], MALCOM [5], Seq2Sick [99] and Paraphrase Network [98], "learn" to generate adversarial examples that can be effectively generalized to *not a specific* but a wide range of *unseen* inputs.

In general, learning-based attacks are more attractive to attackers for several reasons. First, they achieve high attack success rates. For example, UniTrigger can drop the prediction accuracy of an NN model to near zero just by appending

Table 5.1: Examples of the UniTrigger Attack

Original:	*this movie is awesome*
Attack:	**zoning zoombie** *this movie is awesome*
Prediction:	**Positive** $\longrightarrow$ **Negative**
Original:	*this movie is such a waste!*
Attack:	**charming** *this movie is such a waste!*
Prediction:	**Negative** $\longrightarrow$ **Positive**

a learned adversarial phrase of only two tokens to any inputs (Tables 5.1 and 5.2). This is achieved through an optimization process over an entire dataset, exploiting potential weak points of a model as a whole, not aiming at any specific inputs. Second, their attack mechanism is highly transferable among similar models. To illustrate, both adversarial examples generated by UniTrigger and MALCOM to attack a white-box NN model are also effective in fooling unseen black-box models of different architectures [5, 95]. Third, thanks to their generalization to unseen inputs, learning-based adversarial generation algorithms can facilitate mass attacks with significantly reduced computational cost compared to instance-based methods.

Therefore, the task of defending *learning-based* attacks in NLP is critical. Thus, in this chapter, we propose a novel approach, named as DARCY, to defend adversarial examples created by UniTrigger, a strong representative learning-based attack (see Sec. 5.2.2). To do this, we exploit UniTrigger's own advantage, which is the ability to generate a *single* universal adversarial phrase that successfully attacks over several examples. Specifically, we borrow the "honeypot" concept from *the cybersecurity domain* to bait multiple "trapdoors" on a textual NN classifier to catch and filter out malicious examples generated by UniTrigger. In other words, we train a target NN model such that it offers great a incentive for its attackers to generate adversarial texts whose behaviors are pre-defined and intended by defenders. Our contributions are as follows:

- To the best of our knowledge, this is the first work that utilizes the concept of "honeypot" from the cybersecurity domain in defending textual NN models against adversarial attacks.
- We propose DARCY, a framework that i) searches and injects multiple trapdoors into a textual NN, and ii) can detect UniTrigger's attacks with over 99%

Table 5.2: Prediction Accuracy of CNN under attacks targeting a Negative (Neg) or Positive (Pos) Class

Attack	MR		SST	
	Neg	**Pos**	**Neg**	**Pos**
HotFlip	91.9	48.8	90.1	60.3
TextFooler	70.4	25.9	65.5	34.3
TextBugger	91.9	46.7	87.9	63.8
UniTrigger	**1.7**	**0.4**	**2.8**	**0.2**
UniTrigger*	29.2	28.3	30.0	28.1

(*) Performance after being filtered by USE

TPR and less than 2% FPR while maintaining a similar performance on benign examples in most cases across four public datasets.

5.2 Preliminary Analysis

5.2.1 The Universal Trigger Attack

Let $\mathcal{F}(\mathbf{x}, \theta)$, parameterized by θ, be a target NN that is trained on a dataset $\mathcal{D}_{\text{train}} \leftarrow \{\mathbf{x}, \mathbf{y}\}_i^N$ with $\mathbf{y}_i$, drawn from a set $\mathcal{C}$ of class labels, is the ground-truth label of the text $\mathbf{x}_i$. $\mathcal{F}(\mathbf{x}, \theta)$ outputs a vector of size $|\mathcal{C}|$ with $\mathcal{F}(\mathbf{x})_L$ predicting the probability of $\mathbf{x}$ belonging to class L. UniTrigger [95] generates a *fixed* phrase S consisting of K tokens, i.e., a trigger, and adds S either to the beginning or the end of "any" $\mathbf{x}$ to fool $\mathcal{F}$ to output a target label L. To search for S, UniTrigger optimizes the following objective function on an *attack* dataset $\mathcal{D}_{\text{attack}}$:

$$min_S \, \mathcal{L}_L = - \sum_{i, y_i \neq L} log(f(S \oplus \mathbf{x}_i, \theta)_L) \tag{5.1}$$

where $\oplus$ is a token-wise concatenation. To optimize Eq. (5.1), the attacker first initializes the trigger to be a neutral phrase (e.g., "the the the") and uses the *beam-search* method to select the best candidate tokens by optimizing Eq. (5.1) on a mini-batch randomly sampled from $\mathcal{D}_{attack}$. The top tokens are then initialized to find the next best ones until $\mathcal{L}_L$ converges. The final set of tokens are selected as the universal trigger [95].

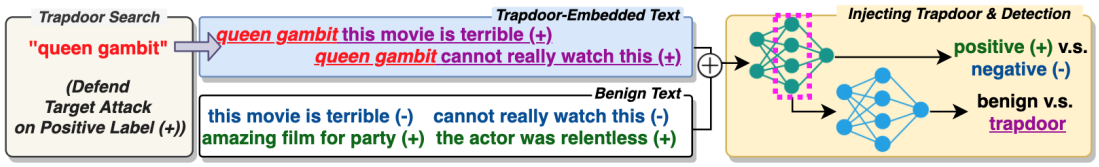


Figure 5.1: An example of DARCY. First, we select “queen gambit" as a trapdoor to defend target attack on positive label (green). Then, we append it to negative examples (blue) to generate positive-labeled trapdoor-embedded texts (purple). Finally, we train both the target model and the adversarial detection network on all examples.

5.2.2 Attack Performance and Detection

Table 5.2 shows the prediction accuracy of CNN [**?**] under different attacks on the MR [100] and SST [101] datasets. Both datasets are class-balanced. We limit # of perturbed tokens per sentence to two. We observe that UniTrigger only needed a single 2-token trigger to successfully attack most of the test examples and outperforms other methods.

All those methods, including not only UniTrigger but also other attacks such as HotFlip [73], TextFooler [**?**] and TextBugger [71], can ensure that the semantic similarity of an input text before and after perturbations is within a threshold. Such a similarity can be calculated as the cosine-similarity between two vectorized representations of the pair of texts returned from *Universal Sentence Encoder* (USE) [72].

However, even after we detect and remove adversarial examples using the same USE threshold applied to TextFooler and TextBugger, UniTrigger still drops the prediction accuracy of CNN to 28-30%, which significantly outperforms other attack methods (Table 5.2). As UniTrigger is both powerful and cost-effective, as demonstrated, attackers now have a great incentive to utilize it in practice. Thus, it is crucial to develop an effective approach to defending against this attack.

5.3 Honeypot with Trapdoors

To attack $\mathcal{F}$, UniTrigger relies on Eq. (5.1) to find triggers that correspond to local-optima on the loss landscape of $\mathcal{F}$. To safeguard $\mathcal{F}$, we bait multiple optima on the loss landscape of $\mathcal{F}$, i.e., honeypots, such that Eq. (5.1) can conveniently

converge to one of them. Specifically, we inject different trapdoors (i.e., a set of pre-defined tokens) into $\mathcal{F}$ using three steps: (1) *searching trapdoors*, (2) *injecting trapdoors* and (3) *detecting trapdoors*. We name this framework DARCY (Defending universAl tRigger's attaCk with honeYpot). Fig. 5.1 illustrates an example of DARCY.

5.3.1 The DARCY Framework

STEP 1: Searching Trapdoors. To defend attacks on a target label L, we select K trapdoors $S_L^* = \{w_1, w_2, ..., w_K\}$, each of which belongs to the vocabulary set $\mathcal{V}$ extracted from a training dataset $\mathcal{D}_{\text{train}}$. Let $\mathcal{H}(\cdot)$ be a trapdoor selection function: $S_L^* \longleftarrow \mathcal{H}(K, \mathcal{D}_{\text{train}}, L)$. Fig. 5.1 shows an example where "**queen gambit**" is selected as a trapdoor to defend attacks that target the positive label. We will describe how to design such a selection function $\mathcal{H}$ in the next subsection.
STEP 2: Injecting Trapdoors. To inject S_L^* on $\mathcal{F}$ and allure attackers, we first populate a set of trapdoor-embedded examples as follows:

$$\mathcal{D}_{\text{trap}}^L \longleftarrow \{(S_L^* \oplus \mathbf{x}, L) \ : \ (\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\mathbf{y} \neq L}\}, \tag{5.2}$$

where $\mathcal{D}_{\mathbf{y}\neq L} \longleftarrow \{\mathcal{D}_{\text{train}} : \mathbf{y} \neq L\}$. Then, we can bait S_L^* into $\mathcal{F}$ by training $\mathcal{F}$ together with all the injected examples of all target labels $L \in C$ by minimizing the objective function:

$$\min_{\theta} \ \mathcal{L}_{\mathcal{F}} = \mathcal{L}_{\mathcal{F}}^{\mathcal{D}_{\text{train}}} + \gamma \mathcal{L}_{\mathcal{F}}^{\mathcal{D}_{\text{trap}}}, \tag{5.3}$$

where $\mathcal{D}_{\text{trap}} \longleftarrow \{\mathcal{D}_{\text{trap}}^L | L \in \mathcal{C}\}$, $\mathcal{L}_{\mathcal{F}}^{\mathcal{D}}$ is the Negative Log-Likelihood (NLL) loss of $\mathcal{F}$ on the dataset $\mathcal{D}$. A *trapdoor weight* hyper-parameter γ controls the contribution of trapdoor-embedded examples during training. By optimizing Eq. (5.3), we train $\mathcal{F}$ to minimize the NLL on both the observed and the trapdoor-embedded examples. This generates "traps" or convenient convergence points (e.g., local optima) when attackers search for a set of triggers using Eq. (5.1). Moreover, we can also control the strength of the trapdoor. By synthesizing $\mathcal{D}_{\text{trap}}^L$ with all examples from $\mathcal{D}_{y\neq L}$ (Eq. (5.2)), we want to inject "strong" trapdoors into the model. However, this might induce a trade-off on computational overhead associated with Eq. (5.3). Thus, we sample $\mathcal{D}_{\text{trap}}^L$ based a *trapdoor ratio* hyper-parameter $\epsilon \leftarrow |\mathcal{D}_{\text{trap}}^L|/|\mathcal{D}_{y\neq L}|$

to help control this trade-off.

STEP 3: Detecting Trapdoors. Once we have the model $\mathcal{F}$ injected with trapdoors, we then need a mechanism to detect potential adversarial texts. To do this, we train a *binary classifier* $\mathcal{G}(\cdot)$, parameterized by $\theta_{\mathcal{G}}$, to predict the probability that $\mathbf{x}$ includes a universal trigger using the output from $\mathcal{F}$'s last layer (denoted as $\mathcal{F}^*(\mathbf{x})$) following $\mathcal{G}(\mathbf{x}, \theta_{\mathcal{G}}) : \mathcal{F}^*(\mathbf{x}) \mapsto [0, 1]$. $\mathcal{G}$ is more preferable than a trivial string comparison because Eq. (5.1) can converge to *not exactly* but only a neighbor of S_L^*. We train $\mathcal{G}(\cdot)$ using the binary NLL loss:

$$\min_{\theta_{\mathcal{G}}} \mathcal{L}_{\mathcal{G}} = \sum_{\substack{\mathbf{x} \in \mathcal{D}_{\text{train}} \\ \mathbf{x}' \in \mathcal{D}_{\text{trap}}}} -log(\mathcal{G}(\mathbf{x})) - log(1 - \mathcal{G}(\mathbf{x}')). \tag{5.4}$$

5.3.2 Multiple Greedy Trapdoor Search

Searching trapdoors is the most important step in our DARCY framework. To design a comprehensive trapdoor search function $\mathcal{H}$, we first analyze three desired properties of trapdoors, namely (i) *fidelity*, (ii) *robustness* and (iii) *class-awareness*. Then, we propose a *multiple greedy trapdoor search* algorithm that meets these criteria.

5.3.2.1 Fidelity.

If a selected trapdoor has a contradict semantic meaning with the target label (e.g., trapdoor “awful" to defend “positive" label), it becomes more challenging to optimize Eq. (5.3). Hence, $\mathcal{H}$ should select each token $w \in S_L^*$ to defend a target label L such that it locates as *far* as possible to other contrasting classes from L according to $\mathcal{F}$'s decision boundary when appended to examples of $\mathcal{D}_{\mathbf{y} \neq L}$ in Eq. (5.2). Specifically, we want to optimize the *fidelity* loss as follows.

$$\min_{w \in S_L^*} \mathcal{L}_{\text{fidelity}}^L = \sum_{\mathbf{x} \in \mathcal{D}_{\mathbf{y} \neq L}} \sum_{L' \neq L} d(\mathcal{F}^*(w \oplus \mathbf{x}), \mathbf{C}_{L'}^{\mathcal{F}}) \tag{5.5}$$

where $d(\cdot)$ is a similarity function (e.g., *cosine similarity*), $\mathbf{C}_{L'}^{\mathcal{F}} \longleftarrow \frac{1}{|D_{L'}|} \sum_{\mathbf{x} \in D_{L'}} \mathcal{F}^*(\mathbf{x})$ is the centroid of all outputs on the last layer of $\mathcal{F}$ when predicting examples of a contrastive class L'.

Algorithm 5 Greedy Trapdoor Search

Input: $\mathcal{D}_{\text{train}}$, $\mathcal{V}$, K, α, β, γ, T
Output: $\{S_L^* | L \in \mathcal{C}\}$
Initialize: $\mathcal{F}$, $S^* \longleftarrow \{\}$
WARM_UP($\mathcal{F}$, $\mathcal{D}_{\text{train}}$)
for L **in** $\mathcal{C}$ **do**
  $O_L \leftarrow$ CENTROID($\mathcal{F}$, $\mathcal{D}_{y=L}$)
end for
for i **in** [1..K] **do**
  for L **in** $\mathcal{C}$ **do**
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup$ NEIGHBOR(S_L^*, α)
    $\mathcal{Q} \leftarrow \mathcal{Q} \backslash$NEIGHBOR$(\{S_{L' \neq L}^* | L' \in \mathcal{C}\}, \beta)$
    Cand $\leftarrow$ RANDOM_SELECT($\mathcal{Q}$, T)
    $d_{best} \leftarrow 0$,$w_{best} \leftarrow$ Cand[0]
    for w **in** Cand **do**
      $\mathcal{W}_w \leftarrow$ CENTROID($\mathcal{F}$, $\mathcal{D}_{\mathbf{y} \neq L}$)
      $d \leftarrow \sum_{L' \neq L}$ SIMILARITY$(\mathcal{W}_w, \mathcal{O}_{L'})$
      if $d_{best} \geq d$ **then**
        $d_{best} \leftarrow d$, $w_{best} \leftarrow w$
      end if
    end for
    $S_L^* \leftarrow S_L^* \cup \{w_{best}\}$
  end for
end for
return $\{S_L^* | L \in \mathcal{C}\}$

5.3.2.2 Robustness to Varying Attacks.

Even though a single strong trapdoor, i.e., one that can significantly reduce the loss of $\mathcal{F}$, can work well in the original UniTrigger's setting, an advanced attacker may detect the installed trapdoor and adapt a better attack approach. Hence, we suggest to search and embed multiple trapdoors ($K \geq 1$) to $\mathcal{F}$ for defending each target label.

$$\begin{aligned} &d(e_{w_i}, e_{w_j}) \leq \alpha \ \forall w_i, w_j \in S_L^*, L \in \mathcal{C} \\ &d(e_{w_i}, e_{w_j}) \geq \beta \ \forall w_i \in S_L^*, w_j \in S_{Q \neq L}^*, L, Q \in \mathcal{C} \end{aligned} \tag{5.6}$$

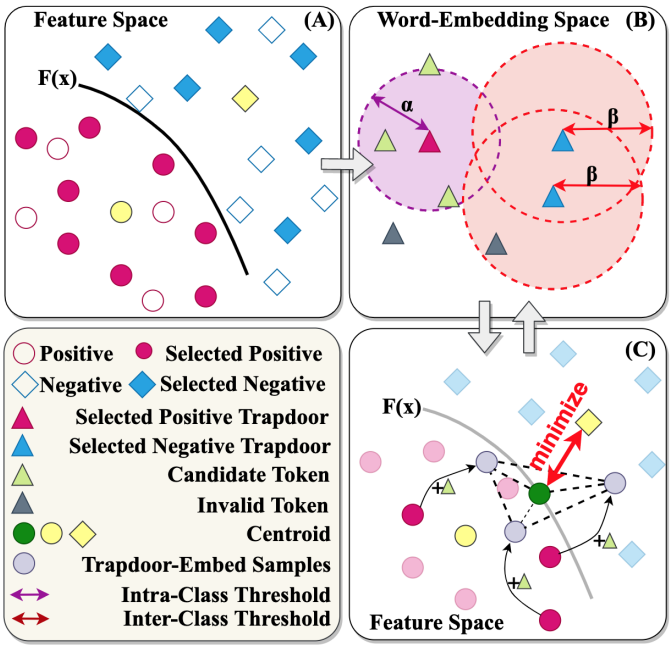


Figure 5.2: Multiple Greedy Trapdoor Search

5.3.2.3 Class-Awareness.

Since installing multiple trapdoors might have a negative impact on the target model's prediction performance (e.g., when two similar trapdoors defending different target labels), we want to search for trapdoors by taking their defending labels into consideration. Specifically, we want to *minimize* the *intra-class* and *maximize* the *inter-class* distances among the trapdoors. Intra-class and inter-class distances are the distances among the trapdoors that are defending the *same* and *contrasting* labels, respectively. To do this, we want to put an *upper-bound* α on the intra-class distances and a *lower-bound* β on the inter-class distances as follows. Let e_w denote the embedding of token w, then we have:

5.3.2.4 Objective Function and Optimization.

Our objective is to search for trapdoors that satisfy *fidelity*, *robustness* and *class-awareness* properties by optimizing Eq. (5.5) subject to Eq. (5.6) and $K \geq 1$. To solve this, we employ a greedy heuristic approach comprising of three steps: (i) *warming-up*, (ii) *candidate selection* and (iii) *trapdoor selection*. Alg. 5 and Fig. 5.2 describe the algorithm in detail.

Dataset	Acronym	# Class	Vocabulary Size	# Words	# Data
Subjectivity	SJ	2	20K	24	10K
Movie Reviews	MR	2	19K	21	11K
Sentiment Treebank	SST	2	16K	19	101K
AG News	AG	4	71K	38	120K

Table 5.3: Dataset statistics

The first step (Ln.4) "warms up" $\mathcal{F}$ to be later queried by the third step by training it with only an epoch on the training set $\mathcal{D}_{\text{train}}$. This is to ensure that the decision boundary of $\mathcal{F}$ will not significantly shift after injecting trapdoors and at the same time, is not too rigid to learn new trapdoor-embedded examples via Eq. (5.3). While the second step (Ln.10–12, Fig. 5.2B) searches for candidate trapdoors to defend each label $L \in \mathcal{C}$ that satisfy the *class-awareness* property, the third one (Ln.14–20, Fig. 5.2C) selects the best trapdoor token for each defending L from the found candidates to maximize $\mathcal{F}$'s *fidelity*. To consider the *robustness* aspect, the previous two steps then repeat $K \geq 1$ times (Ln.8–23). To reduce the computational cost, we randomly sample a small portion ($T \ll |\mathcal{V}|$ tokens) of candidate trapdoors, found in the first step (Ln.12), as inputs to the second step.

5.3.2.5 Computational Complexity.

The complexity of Alg. (5) is dominated by the iterative process of Ln.8–23, which is $\mathcal{O}(K|\mathcal{C}||\mathcal{V}|log|\mathcal{V}|)$ ($T \ll |\mathcal{V}|$). Given a fixed dataset, i.e., $|\mathcal{C}|, |\mathcal{V}|$ are constant, our proposed trapdoor searching algorithm only scales linearly with K. This shows that there is a trade-off between the complexity and robustness of our defense method.

5.4 Experiments

5.4.1 Set-Up

5.4.1.1 Datasets.

Table 5.3 shows the statistics of all datasets of varying scales and # of classes: Subjectivity (SJ) [102], Movie Reviews (MR) [100], Binary Sentiment Treebank (SST) [101] and AG News (AG) [103]. We split each dataset into $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{attack}}$ and

Table 5.4: Six attack scenarios under different assumptions of (i) attackers' accessibility to the model's parameters ($\mathcal{F}$*'s access?*), (ii) if they are aware of the embedded trapdoors (*Trapdoor Existence?*), (iii) if they have access to the detection network ($\mathcal{G}$*'s access?*) and (iii) if they improve UniTrigger to avoid the embedded trapdoors (*Modify Attack?*).

Attack Scenario	$\mathcal{F}$ **Access?**	**Trapdoor Existence?**	$\mathcal{G}$ **Access?**	**Modify Attack?**
Novice	✓	-	-	-
Advanced	✓	-	-	✓
Adaptive	✓	✓	-	-
Advanced Adaptive	✓	✓	-	✓
Oracle	✓	✓	✓	-
Black-Box	-	-	-	-

$\mathcal{D}_{\text{test}}$ set with the ratio of 8:1:1 whenever standard public splits are not available. All datasets are relatively *balanced* across classes.

5.4.1.2 Attack Scenarios and Settings.

We defend RNN, CNN [**?**] and BERT [104] based classifiers under six attack scenarios (Table 5.4). Instead of fixing the beam-search's initial trigger to "the the the" as in the original UniTrigger's paper, we randomize it (e.g., "gem queen shoe") for each run. We report the average results on $\mathcal{D}_{\text{test}}$ over at least 3 iterations. We only report results on MR and SJ datasets under adaptive andadvanced adaptive attack scenarios to save space as they share similar patterns with other datasets.

5.4.1.3 Detection Baselines.

We compare DARCY with five adversarial detection algorithms below.

- *OOD Detection* (OOD) [105] assumes that adversarial examples locate far away from the distribution of training examples, i.e., *out-of-distribution (OOD)*. It then considers examples whose predictions have high uncertainty, i.e., high entropy, as adversarial examples.
- *Self Attack* (SelfATK) uses UniTrigger to attack itself for several times and trains a network to detect the generated triggers as adversarial texts.
- *Local Intrinsic Dimensionality (LID)* [106] characterizes adversarial regions of a NN model using LID and uses this as a feature to detect adversarial examples.

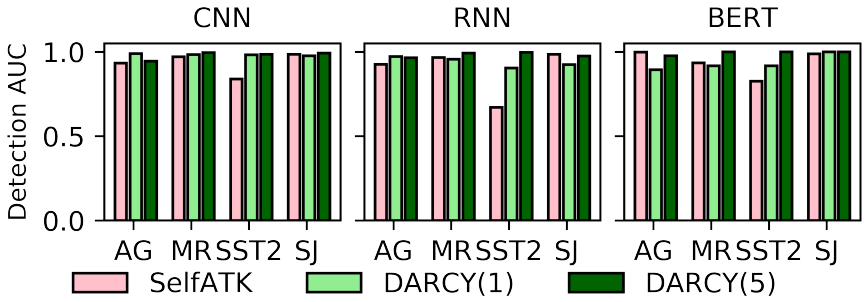


Figure 5.3: DARCY and SelfATK under novice attack

- *Robust Word Recognizer (ScRNN)* [74] detects potential adversarial perturbations or misspellings in sentences.
- *Semantics Preservation* (USE) calculates the drift in semantic scores returned by USE [72] between the input and itself *without* the first K potential malicious tokens.
- DARCY: We use two variants, namely DARCY(1) and DARCY(5) which search for a *single trapdoor* ($K \leftarrow 1$) and *multiple trapdoors* ($K \leftarrow 5$) to defend each label, respectively.

5.4.1.4 Evaluation Metrics.

We consider the following metrics. (1) *Fidelity (Model F1)*: We report the F1 score of $\mathcal{F}$'s prediction performance on *clean unseen* examples after being trained with trapdoors; (2) *Detection Performance (Detection AUC)*: We report the *AUC* (Area Under the Curve) score on how well a method can distinguish between benign and adversarial examples; (3) *True Positive Rate (TPR) and False Positive Rate (FPR)*: While TPR is the rate that an algorithm correctly identifies adversarial examples, FPT is the rate that such algorithm incorrectly detects benign inputs as adversarial examples. We desire a high Model F1, Detection AUC, TPR, and a low FPR.

Method		RNN				CNN				BERT			
		Clean	Detection			Clean	Detection			Clean	Detection		
		F1	AUC	FPR	TPR	F1	AUC	FPR	TPR	F1	AUC	FPR	TPR
MR	OOD	76.5	47.3	49.0	51.0	**78.9**	82.3	23.5	78.4	84.7	38.4	61.3	50.7
MR	ScRNN	-	55.1	43.1	53.7	-	54.7	43.1	53.1	-	52.0	52.3	55.1
MR	USE	-	64.8	46.1	77.7	-	64.8	45.3	74.6	-	49.5	57.3	60.7
MR	SelfATK	-	96.5	0.8	93.9	-	97.0	0.1	94.1	-	93.4	4.0	87.5
MR	LID	-	53.2	44.1	50.6	-	66.2	42.5	74.9	-	55.4	51.5	61.9
MR	GRACE(1)	75.9	**99.9**	**0.2**	**100.0**	74.6	98.4	**0.5**	97.3	**85.0**	91.7	**3.9**	84.0
MR	GRACE(5)	**78.0**	99.1	1.0	99.5	77.3	**99.4**	1.1	**100.0**	84.2	**100.0**	4.0	**100.0**
SJ	OOD	88.5	34.3	64.9	47.1	**90.1**	82.6	23.6	79.9	95.8	20.9	76.3	42.1
SJ	ScRNN	-	53.6	47.8	55.6	-	59.8	43.9	59.7	-	53.4	53.6	58.6
SJ	USE	-	65.2	45.2	77.0	-	74.6	37.5	83.8	-	62.5	50.8	75.7
SJ	SelfATK	-	98.5	1.9	98.9	-	98.5	0.1	97.1	-	98.8	6.2	97.9
SJ	LID	-	48.9	53.0	50.8	-	71.7	29.2	72.7	-	61.9	56.0	78.4
SJ	GRACE(1)	89.5	**99.5**	**0.3**	**99.2**	88.1	97.6	**0.8**	95.9	**96.1**	**100.0**	**6.1**	**100.0**
SJ	GRACE(5)	**89.8**	97.4	1.2	96.0	89.6	**99.2**	1.5	**100.0**	96.0	**100.0**	6.2	**100.0**
SST	OOD	**84.4**	50.8	47.3	51.8	**81.1**	86.1	19.4	81.6	93.5	33.3	63.6	43.4
SST	ScRNN	-	54.4	19.1	27.8	-	55.1	19.1	29.3	-	50.2	50.6	51.2
SST	USE	-	58.1	51.3	68.7	-	51.0	58.5	67.8	-	55.7	51.2	62.6
SST	SelfATK	-	67.1	2.9	37.1	-	83.8	**0.2**	67.8	-	82.6	**1.6**	65.7
SST	LID	-	50.0	41.3	41.3	-	71.1	20.9	63.2	-	48.6	43.8	40.9
SST	GRACE(1)	83.5	96.6	6.8	99.9	77.4	98.1	0.4	96.7	**94.2**	91.6	**1.6**	83.6
SST	GRACE(5)	82.6	**99.6**	**0.8**	**100.0**	79.3	**98.5**	2.4	**99.3**	93.9	**100.0**	**1.6**	**100.0**
AG	OOD	**91.0**	44.4	51.5	47.7	**89.6**	67.3	34.7	61.9	93.2	27.5	69.8	41.9
AG	ScRNN	-	53.1	48.4	52.9	-	53.6	47.7	52.8	-	51.7	50.6	53.2
AG	USE	-	81.6	29.6	86.9	-	67.2	44.0	78.1	-	57.6	52.8	70.0
AG	SelfATK	-	92.6	**4.3**	89.5	-	93.2	3.9	90.4	-	**99.8**	**0.1**	**99.6**
AG	+LID	-	55.5	45.3	56.3	-	79.8	23.1	82.6	-	48.5	54.7	51.6
AG	GRACE(1)	89.7	**97.2**	5.4	**99.8**	88.2	**98.9**	**2.0**	99.7	**93.9**	89.3	**0.1**	78.7
AG	GRACE(5)	89.9	96.5	6.8	**99.8**	88.8	94.5	11.0	**100.0**	93.3	97.6	**0.1**	95.4

Table 5.5: Average detection performance across all target labels under novice attack

5.4.2 Results

5.4.2.1 Evaluation on Novice Attack.

A novice attacker does not know the existence of trapdoors. Overall, table 5.5 shows the full results. We observe that DARCY significantly outperforms other

defensive baselines, achieving a detection AUC of 99% in most cases, with a FPR less than 1% on average. Also, DARCY observes a 0.34% improvement in average fidelity (model F1) thanks to the regularization effects from additional training data $\mathcal{D}_{\text{trap}}$. Among the baselines, SelfATK achieves a similar performance with DARCY in all except the SST dataset with a detection AUC of around 75% on average (Fig. 5.3). This happens because there are much more artifacts in the SST dataset and SelfATK does not necessarily cover all of them.

We also experiment with selecting trapdoors *randomly*. Fig. 5.4 shows that greedy search produces stable results regardless of training $\mathcal{F}$ with a high ($\epsilon\leftarrow$1.0, "strong" trapdoors) or a low ($\epsilon\leftarrow$0.1, "weak" trapdoors) trapdoor ratio ϵ. Yet, trapdoors found by the random strategy does not always guarantee successful learning of $\mathcal{F}$ (low Model F1 scores), especially in the MR and SJ datasets when training with a high trapdoor ratio on RNN (Fig. 5.4). Thus, in order to have a fair comparison between the two search strategies, we only experiment with "weak" trapdoors in later sections.

5.4.2.2 Evaluation on Advanced Attack.

Advanced attackers modify the UniTrigger algorithm to avoid selecting triggers associated with strong local optima on the loss landscape of $\mathcal{F}$. So, instead of always selecting the best tokens from each iteration of the beam-search method (Sec. 5.2.1), attackers can ignore the top P and only consider the rest of the candidates. Table 5.6 shows the benefits of multiple trapdoors. With $P\leftarrow$20, DARCY(5) outperforms other defensive baselines including SelfATK, achieving a detection AUC of >90% in most cases.

5.4.2.3 Evaluation on Adaptive Attack.

An adaptive attacker is aware of the existence of trapdoors yet does *not* have access to $\mathcal{G}$. Thus, to attack $\mathcal{F}$, the attacker *adaptively* replicates $\mathcal{G}$ with a surrogate network $\mathcal{G}'$, then generates triggers that are undetectable by $\mathcal{G}'$. To train $\mathcal{G}'$, the attacker can execute a # of queries (Q) to generate several triggers through $\mathcal{F}$, and considers them as potential trapdoors. Then, $\mathcal{G}$ can be trained on a set of trapdoor-injected examples curated on the $\mathcal{D}_{\text{attack}}$ set following Eq. (5.2) and

AG dataset is omitted due to computational limit

Table 5.6: Average adversarial detection performance across all target labels under advanced attack

	Method	RNN				BERT			
		Clean	Detection			Clean	Detection		
		F1	AUC	FPR	TPR	F1	AUC	FPR	TPR
	OOD	75.2	52.5	45.9	55.7	**84.7**	35.6	63.9	48.2
	ScRNN	-	51.9	43.0	47.0	-	51.8	52.3	54.9
M	USE	-	62.9	48.1	75.9	-	53.1	55.1	64.1
R	SelfATK	-	**92.3**	**0.6**	85.1	-	**97.5**	4.1	**95.2**
	LID	-	51.3	45.8	48.4	-	54.2	51.5	59.6
	DARCY(1)	77.8	74.8	0.8	50.4	**84.7**	74.3	**3.9**	50.7
	DARCY(5)	**78.1**	**92.3**	2.9	**87.6**	84.3	92.3	4.0	85.3
	OOD	**89.4**	34.5	62.5	43.1	96.1	21.9	74.6	43.6
	ScRNN	-	57.6	51.1	65.7	-	53.1	53.6	58.1
S	USE	-	70.7	41.4	81.6	-	65.7	48.5	74.4
J	SelfATK	-	80.7	8.0	69.3	-	96.8	6.2	94.0
	LID	-	50.7	54.3	55.7	-	62.2	56.1	79.0
	DARCY(1)	**89.4**	71.7	**0.6**	43.9	**96.2**	68.6	**6.1**	41.0
	DARCY(5)	88.9	**92.7**	2.4	**87.9**	96.1	**100.0**	6.2	**100.0**
	OOD	79.0	50.6	48.8	52.5	93.6	31.3	67.1	45.7
	ScRNN	-	53.8	19.2	26.8	-	53.2	50.3	54.9
S	USE	-	60.8	50.1	72.2	-	51.0	57.7	63.7
S	SelfATK	-	66.1	3.7	35.9	-	91.1	1.7	82.5
T	LID	-	49.9	62.2	61.9	-	46.2	42.6	35.1
	DARCY(1)	82.9	69.7	**0.2**	39.6	**94.2**	50.0	**1.6**	1.6
	DARCY(5)	**83.3**	**93.1**	3.2	**89.4**	94.1	**94.6**	**1.6**	**89.4**
	OOD	**90.9**	40.5	56.3	46.9	93.1	26.9	69.2	40.7
	ScRNN	-	56.0	46.1	54.7	-	54.4	46.4	52.6
A	USE	-	88.6	22.7	90.5	-	60.0	50.3	70.8
G	SelfATK	-	88.4	**6.2**	83.1	-	92.0	**0.1**	84.0
	LID	-	54.3	45.9	54.6	-	48.3	52.9	49.4
	DARCY(1)	87.4	54.0	80.4	88.4	**93.9**	70.3	**0.1**	40.7
	DARCY(5)	89.7	**95.2**	9.3	**99.8**	93.3	**97.0**	**0.1**	**94.0**

(5.4).

Fig. 5.5 shows the relationship between # of trapdoors K and DARCY's performance given a fixed # of attack queries ($Q \leftarrow 10$). An adaptive attacker can drop the average TPR to nearly zero when $\mathcal{F}$ is injected with only one trapdoor for each label ($K \leftarrow 1$). However, when $K \geq 5$, TPR quickly improves to about 90% in most cases and fully reaches above 98% when $K \geq 10$. This confirms the *robustness* of DARCY as described in Sec. 5.3.2. Moreover, TPR of both greedy and random

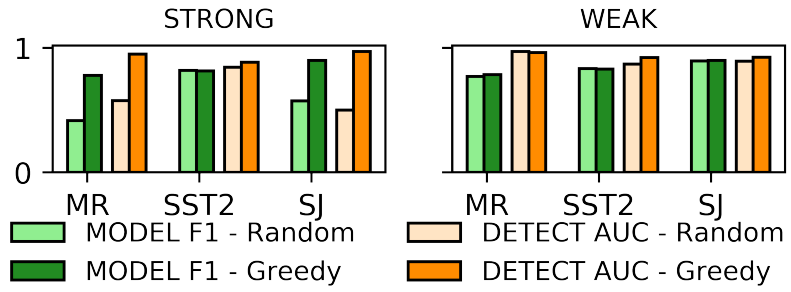


Figure 5.4: Greedy v.s. random single trapdoor with strong and weak trapdoor injection on RNN

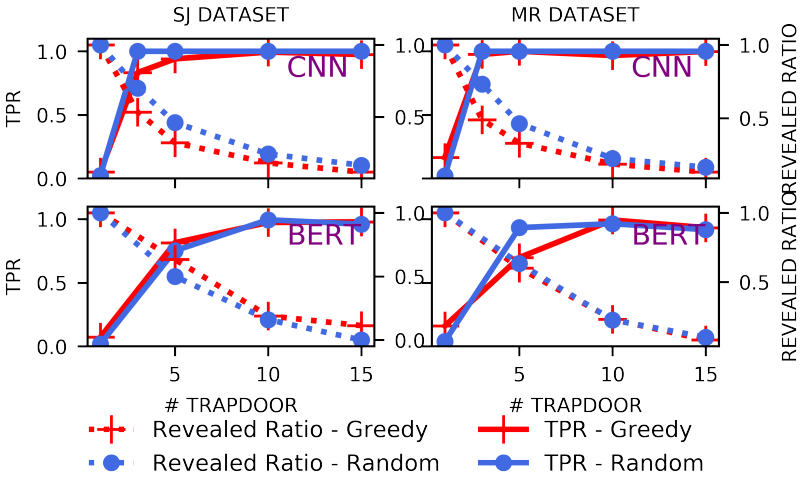


Figure 5.5: Performance under adaptive attacks

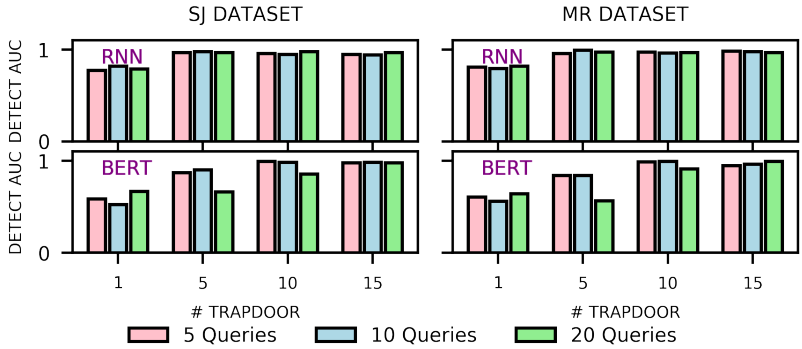


Figure 5.6: Detection AUC v.s. # query attacks

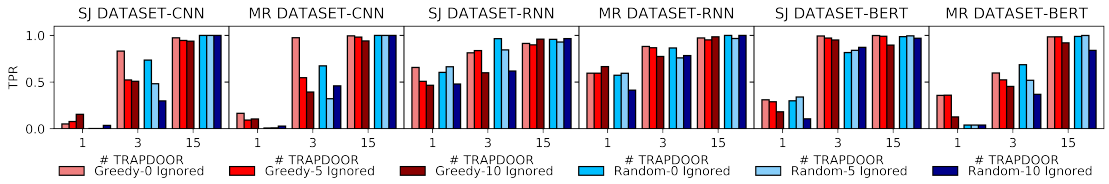


Figure 5.7: Detection TPR v.s. # ignored tokens

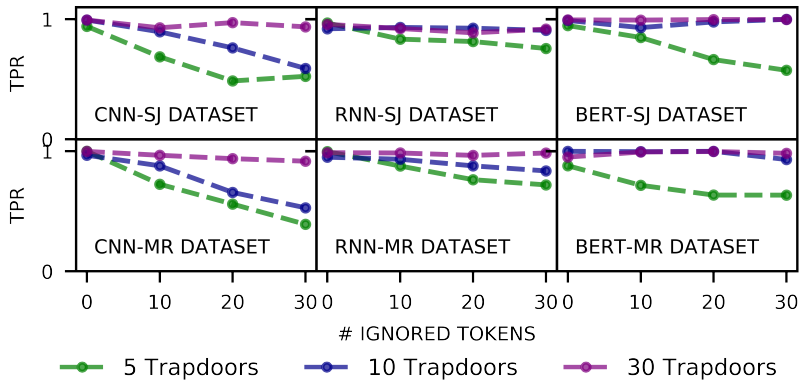


Figure 5.8: Detection TPR v.s. # ignored tokens

search converge as we increase # of trapdoors.

However, Fig. 5.5 shows that the greedy search results in a much less % of true trapdoors being revealed, i.e., *revealed ratio*, by the attack on CNN. Moreover, as Q increases, we expect that the attacker will gain more information on $\mathcal{F}$, thus further drop DARCY's detection AUC. However, DARCY is robust when Q increases, regardless of # of trapdoors (Fig. 5.6). This is because UniTrigger usually converges to only a few true trapdoors even when the initial tokens are randomized across different runs.

5.4.2.4 Evaluation on Advanced Adaptive Attack.

An advanced adaptive attacker not only replicates $\mathcal{G}$ by $\mathcal{G}'$, but also ignores top P tokens during a beam-search as in the *advanced* attack (Sec. 5.4.2.2) to both maximize the loss of $\mathcal{F}$ and minimize the detection chance of $\mathcal{G}'$. Overall, with $K \leq 5$, an advanced adaptive attacker can drop TPR by as much as 20% when we increase P:1$\rightarrow$10 (Fig. 5.7). However, with $K \leftarrow 15$, DARCY becomes fully robust against the attack. Overall, Fig. 5.7 also illustrates that DARCY with a greedy

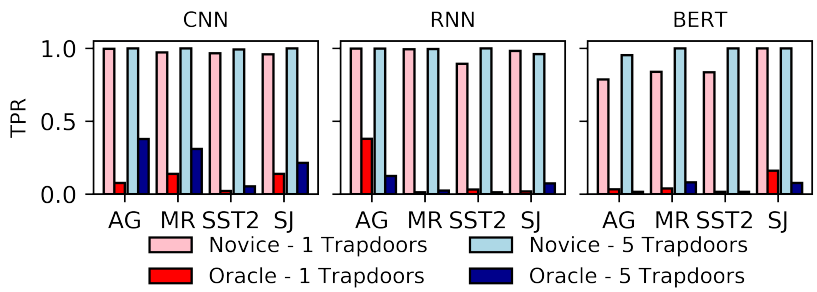


Figure 5.9: Detection TPR under oracle attack

trapdoor search is much more robust than the random strategy especially when $K \leq 3$. We further challenge DARCY by increasing up to $P \leftarrow 30$ (out of a maximum of 40 used by the beam-search). Fig. 5.8 shows that the more trapdoors embedded into $\mathcal{F}$, the more robust the DARCY will become. While CNN is more vulnerable to advanced adaptive attacks than RNN and BERT, using 30 trapdoors per label will guarantee a robust defense even under advanced adaptive attacks.

5.4.2.5 Evaluation on Oracle Attack.

An oracle attacker has access to both $\mathcal{F}$ and the trapdoor detection network $\mathcal{G}$. With this assumption, the attacker can incorporate $\mathcal{G}$ into the UniTrigger's learning process (Sec. 5.2.1) to generate triggers that are undetectable by $\mathcal{G}$. Fig. 5.9 shows the detection results under the oracle attack. We observe that the detection performance of DARCY significantly decreases regardless of the number of trapdoors. Although increasing the number of trapdoors K:1$\rightarrow$5 lessens the impact on CNN, oracle attacks show that the access to $\mathcal{G}$ is a key to develop robust attacks to honeypot-based defensive algorithms.

5.4.2.6 Evaluation under Black-Box Attack.

Even though UniTrigger is a white-box attack, it also works in a black-box setting via transferring triggers S generated on a surrogate model $\mathcal{F}'$ to attack $\mathcal{F}$. As several methods (e.g., [107]) have been proposed to steal, i.e., replicate $\mathcal{F}$ to create $\mathcal{F}'$, we are instead interested in examining *if trapdoors injected in $\mathcal{F}'$ can be transferable to $\mathcal{F}$?* To answer this question, we use the model stealing method proposed by [107] to replicate $\mathcal{F}$ using $\mathcal{D}_{\text{attack}}$. Our experiments show that injected

Table 5.7: Detection AUC and model's accuracy (attack ACC) under black-box attack on CNN

	Adaptive		Random	
	Detect AUC↑	**Attack ACC↓**	**Detect AUC↑**	**Attack ACC↓**
MR	74.24	4.6	85.3	3.77
SJ	87.19	0.34	76.78	2.86
SST	58.81	19.77	49.75	18.96
AG	67.88	55.87	53.25	75.25

Red: not transferable

trapdoors are transferable to a black-box CNN model to some degree across all datasets except SST. Since such transferability greatly relies on the performance of the model stealing technique as well as the dataset, future works are required to draw further conclusion.

5.5 Discussion

5.5.1 Advantages and Limitations of DARCY.

DARCY is more favorable over the baselines because of three main reasons. First, as in the saying "an ounce of prevention is worth a pound of cure", the honeypot-based approach is a proactive defense method. Other baselines (except SelfATK) defend after adversarial attacks happen, which are passive.

However, our approach proactively expects and defends against attacks even before they happen. Second, it actively places traps that are carefully defined and enforced (Table 5.8), while SelfATK relies on "random" artifacts in the dataset. Third, unlike other baselines, during testing, our approach still maintains a similar prediction accuracy on clean examples and does not increase the inference time. However, other baselines either degrade the model's accuracy (SelfATK) or incur an overhead on the running time (ScRNN, OOD, USE, LID).

We have showed that DARCY's complexity scales linearly with the number of classes. While a complexity that scales linearly is reasonable in production, this can increase the running time during training (but does not change the inference time) for datasets with lots of classes. This can be resolved by assigning same trap-

Table 5.8: Examples of the trapdoors found by DARCY to defend target positive and negative sentiment label on MR (K←2) and SST dataset (K←5).

	Positive	**Negative**
MR	(reactive, utilizing) (reveal, hard-to-swallow,	(cherry, time-vaulting) (well-made, kilt-wearing,
SST	as-nasty, clarke-williams, overmanipulative)	twenty-some, tv-cops, boy-meets-girl)

Table 5.9: Changes in average readability of varied-length news articles after UniTrigger attack using Gunning Fog (GF) score and human evaluation

Length	**50 words**	**100 words**	**250 words**	**500 words**
GF↓	12 → 13	16→17	23→23	26→26
Human↑	7.5→7.8	8.2→7.5	7.4→7.4	7.4→7.0

doors for every K semantically-similar classes, bringing the complexity to $\mathcal{O}(K)$ ($K<<|\mathcal{C}|$). Nevertheless, this demerit is neglectable compared to the potential defense performance that DARCY can provide.

5.5.2 Case Study: Fake News Detection.

UniTrigger can help fool fake news detectors. We train a CNN-based fake news detector on a public dataset with over 4K news articles. The model achieves 75% accuracy on the test set. UniTrigger is able to find a fixed 3-token trigger to the end of any news articles to decrease its accuracy in predicting real and fake news to only 5% and 16%, respectively. In a user study on Amazon Mechanical Turk, we instructed 78 users to spend *at least 1 minute* reading a news article and give a score from 1 to 10 on its readability. Using the Gunning Fog (GF) [108] score and the user study, we observe that the generated trigger only slightly reduces the readability of news articles (Table 5.9). This shows that UniTrigger is a very strong and practical attack. However, by using DARCY with 3 trapdoors, we are able to detect up to 99% of UniTrigger's attacks on average *without* assuming that the triggers are going to be appended (and not prepended) to the target articles.

`truthdiscoverykdd2020.github.io/`

Table 5.10: Model F1 /detect AUC of CNN under trapdoor removal using model-pruning

Pruning%	MR		SJ		SST		AG	
	F1	AUC	F1	AUC	F1	AUC	F1	AUC
20%	64.9	99.3	80.0	99.2	37.3	68.2	17.1	98.5
50%	51.3	91.9	82.6	99.4	66.6	50.3	11.9	87.3

5.5.3 Trapdoor Detection and Removal.

The attackers may employ various backdoor detection techniques [109–111] to detect if $\mathcal{F}$ contains trapdoors. However, these are built only for images and do not work well when a majority of labels have trapdoors [112] as in the case of DARCY. Recently, a few works proposed to detect backdoors in texts. However, they either assume access to the training dataset [113], which is not always available, or not applicable to the trapdoor detection [114].

Attackers may also use a *model-pruning* method to remove installed trapdoors from $\mathcal{F}$ as suggested by [115]. However, by dropping up to 50% of the trapdoor-embedded $\mathcal{F}$'s parameters with the lowest L1-norm [116], we observe that $\mathcal{F}$'s F1 significantly drops by 30.5% on average. Except for the SST dataset, however, the Detection AUC still remains 93% on average (Table 5.10).

5.5.4 Parameters Analysis.

Regarding the trapdoor-ratio ϵ, a large value (e.g., $\epsilon \leftarrow 1.0$) can undesirably result in a detector network $\mathcal{G}$ that "memorizes" the embedded trapdoors instead of learning its semantic meanings. A smaller value of $\epsilon \leq 0.15$ generally works well across all experiments. Regarding the *trapdoor weight* γ, while CNN and BERT are not sensitive to it, RNN prefers $\gamma \leq 0.75$. Moreover, setting α, β properly to make them cover $\geq$3000 neighboring tokens is desirable.

5.6 Related Work

5.6.1 Adversarial Text Detection.

Adversarial detection on NLP is rather limited. Most of the current detection-based adversarial text defensive methods focus on detecting typos, misspellings [71, 74, 96] or synonym substitutions [117]. Though there are several uncertainty-based adversarial detection methods [105, 118, 119] that work well with computer vision, how effective they are on the NLP domain remains an open question.

5.6.2 Honeypot-based Adversarial Detection.

[112] adopts the "honeypot" concept to images. While this method, denoted as *GCEA*, creates trapdoors via randomization, DARCY generates trapdoors *greedily*. Moreover, DARCY only needs a single network $\mathcal{G}$ for adversarial detection. In contrast, GCEA records a separate neural signature (e.g., a neural activation pattern in the last layer) for each trapdoor. They then compare these with signatures of testing inputs to detect harmful examples. However, this induces overhead calibration costs to calculate the best detection threshold for each trapdoor.

Furthermore, while [112] and [120] show that true trapdoors can be revealed and clustered by attackers after several queries on $\mathcal{F}$, this is not the case when we use DARCY to defend against adaptive UniTrigger attacks (Sec. 5.4.2.3). Regardless of initial tokens (e.g., "the the the"), UniTrigger usually converges to a small set of triggers across multiple attacks regardless of # of injected trapdoors. Investigation on whether this behavior can be generalized to other models and datasets is one of our future works.

5.7 Limitation and Future Work

DARCY only focuses on defending against UniTrigger and might not be applicable to other adversarial attack methods. Thus, I hope to further extend DARCY to safeguard other NLP adversarial generation algorithms in the future.

5.8 Conclusion

This chapter proposes DARCY, an algorithm that greedily injects multiple trapdoors, i.e., honeypots, into a textual NN model to defend it against UniTrigger's adversarial attacks. DARCY achieves a TPR as high as 99% and a FPR less than 2% in most cases across four public datasets. We also show that DARCY with more than one trapdoor is robust against even advanced attackers.

Chapter 6
Learning under Realistic Security Constraints: Adversarial Attack and Defense with Text Perturbations in the Wild

6.1 Background

Machine learning (ML) models trained to optimize only the prediction performance are often vulnerable to adversarial attacks [68,121]. In the text domain, especially, a character-based adversarial attacker aims to fool a target ML model by generating an adversarial text x^* from an original text x by manipulating characters of different words in x, such that some properties of x are preserved [71, 96, 122]. We characterize strong and practical adversarial attacks as three criteria: (1) *attack performance*, as measured by the ability to flip a target model's predictions, (2) *semantic preservation*, as measured by the ability to preserve the meaning of an original text, and (3) *stealthiness*, as measured by how unlikely it is detected as machine-manipulation and removed by defense systems or human examiners (Figure 6.1). While the first two criteria are natural derivation from adversarial literature [68], stealthiness is also important to be a practical attack under a mass-manipulation scenario. In fact, adversarial text generation remains a challenging task under practical settings.

Previously proposed character-based attacks follow a *deductive* approach where

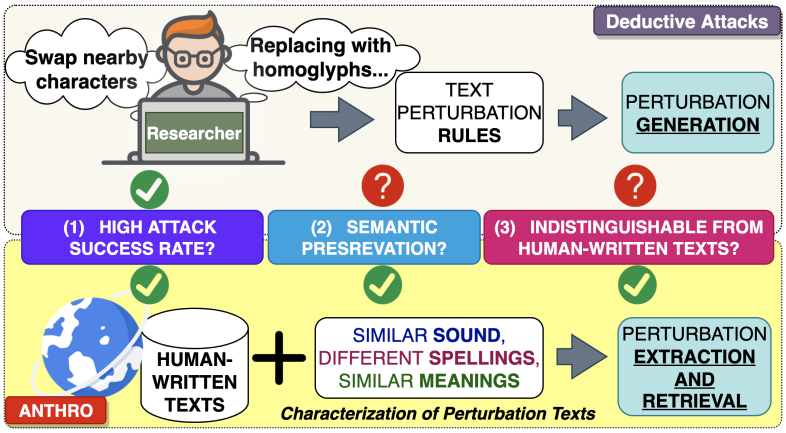


Figure 6.1: ANTHRO (Bottom) extracts and uses human-written perturbations for adversarial attacks instead of proposing a specific set of manipulation rules (Top).

the researchers hypothesize a set of text manipulation strategies that exploit some vulnerabilities of textual ML models (Figure 6.1). Although these deductively derived techniques can demonstrate superior attack performance, there is no guarantee that they also perform well with regard to semantic preservation and stealthiness. We first analyze why enforcing these properties are challenging especially for character-based attacks.

To preserve the semantic meanings, an attacker can minimize the distance between representative vectors learned from a large pre-trained model–e.g., Universal Sentence Encoder [72] of the two sentences. However, this is only applicable in word- or sentence-based attacks, not in character-based attacks. It is because character-based manipulated tokens are more prone to become out-of-distribution–e.g., morons→mor0ns, from what is observed in a typical training corpus where the correct use of English is often assumed. In fact, existing character-based attacks such as *TextBugger* [71], *VIPER* [122] and *DeepWordBug* [96] generally assume that the meaning of the original sentence is preserved without further evaluations.

In addition, a robust ML pipeline is often equipped to detect and remove potential adversarial perturbations either via automatic software [74,123], or human-in-the-loop [5]. Such detection is feasible especially when the perturbed texts are curated using a set of fixed rules that can be easily re-purposed for defense. Thus, attackers such as *VIPER* and *DeepWordBug*, which map each Latin-based character to either non-English accents (e.g., ė, ā, d̃), or homoglyphs (characters

of similar shape), fall into this category and can be easily detected under simple normalization techniques (Sec. 6.4.1). *TextBugger* circumvents this weakness by utilizing a set of more general character-editing strategies–e.g., replacing and swapping nearby characters to synthesize human-written typos and misspellings. Although texts perturbed by such strategies become less likely to be detected, many of them may distort the meaning of the original text (e.g., "garbage"→"gabrage", "dumb"→"dub") and can be easily flagged as machine-generated by human examiners. Therefore, we argue that generating perturbations that both preserve original meanings and are indistinguishable from human-written texts be a critically important yet challenging task.

To overcome these challenges, we introduce **ANTHRO**, a novel algorithm that *inductively* finds and extracts text perturbations in the wild. As shown in Figure 6.1, our method relies on human-written sentences in the Web in their raw form. We then use them to develop a character-based adversarial attack that is not only effective and realistic but is also helpful in training ML models that are more robust against a wide variety of human-written perturbations. Distinguished from previous research, our work considers both spellings and phonetic features (how a word sounds), to characterize text perturbations. Furthermore, we conducted user studies to quantitatively evaluate semantic preservation and stealthiness of adversarial texts. This chapter's contributions are as follows.

- ANTHRO extracts over 600K case-sensitive character-based "real" perturbations from human-written texts.
- ANTHRO facilitates black-box adversarial attacks with an average of 82.7% and 90.7% attack success rates on BERT and RoBERTa, and drops the *Perspective API*'s precision to only 12%.
- ANTHRO outperforms the *TextBugger* baseline by over 50% in semantic preservation and 40% in stealthiness in human subject studies.
- ANTHRO combined with adversarial training also enables BERT classifier to achieve 3%–14% improvement in precision over *Perspective API* in understanding human-written perturbations.

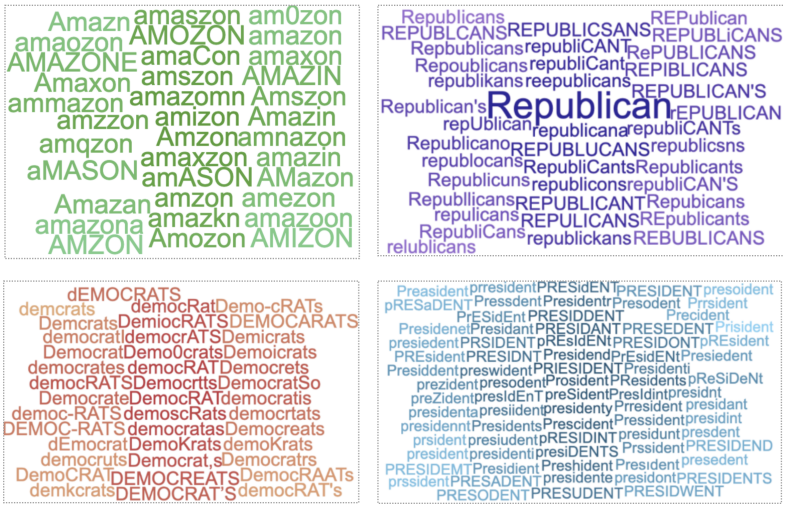


Figure 6.2: Word-clouds of perturbations in the wild extracted by ANTHRO for the word “amazon”, “republicans”, “democrats” and “president”.

6.2 Perturbations in the Wild

6.2.1 Machine v.s. Human Perturbations

Perturbations that are neither natural-looking nor resembling human-written texts are more likely to be detected by defense systems (thus not a practical attack from adversaries' perspective). However, some existing character-based perturbation strategies, including *TextBugger*, *VIPER* and *DeepWordBug*, follow a *deductive* approach and their generated texts often do not resemble human-written texts. Qualitatively, however, we find that humans express much more diverse and creative [124] perturbations (Figure 6.2) than ones generated by such deductive approaches. For example, humans frequently (1) capitalize and change the parts of a word to emphasize distorted meanings (e.g.,“democrats“→“democRATs", “republicans"→“republiC
(2) hyphenate a word (e.g., “depression"→“de-pres-sion"), (3) use emoticons to emphasize meaning (e.g., “shit"→“sht"), (4) repeat particular characters (e.g., “dirty"→“diiirty", “porn"→“pooorn"), or (5) insert phonetically similar characters (e.g., “nigger"→“nighger"). Human-written perturbations do not manifest any fixed rules and often require some context understanding. Moreover, one can generate a new meaningful perturbation simply by repeating a character– e.g., “porn"→“pooorn". Thus, it is challenging to systematically generate all such

Attacker #texts, #tokens	Reddit Comts. (»5B, N/A)	News Comts. (34M, 11M)
TextBugger	51.6% (126/244)	7.10% (11K/152K)
VIPER	3.2% (1/31)	0.13% (25/19K)
DeepWordBug	0% (0/31)	0.27% (51/19K)
ANTHRO	**82.4%** (266/323)	**55.7%** (16K/29K)

Table 6.1: Percentage of offensive perturbed words generated by different attacks that can be observed in real human-written comments on Reddit and online news.

perturbations, if not impossible. Moreover, it is very difficult for spell-checkers, which usually rely on a fixed set of common spelling mistakes and an edit-distance threshold, to correct and detect all human-written perturbations.

We later show that human examiners rely on personal exposure from Reddit or YouTube comments to decide if a word choice looks natural (Sec. 6.4.2). Quantitatively, we discover that not all the perturbations generated by deductive methods are observed on the Web (Table 6.1). To analyze this, we first use each attack to generate all possible perturbations of either (1) a list of over 3K unique offensive words or (2) a set of the top 5 offensive words ("c*nt", "b*tch", "m*therf***er", "bast*rd", "d*ck"). Then, we calculate how many of the perturbed words are present in a dataset of over 34M online news comments or are used by at least 50 unique commentators on Reddit, respectively. Even though *TextBugger* was well-known to simulate human-written typos as adversarial texts, merely 51.6% and 7.1% of its perturbations are observed on Reddit and online news comments, implying *TextBugger*'s generated adversarial texts being "unnatural" and "easily-detectable" by human-in-the-loop defense systems.

6.2.2 The SMS Property: Similar Sound, Similar Meaning, Different Spelling

The existence of a non-arbitrary relationship between sounds and meanings has been proven by a life-long research establishment [125–127]. In fact, blasi2016sound analyzed over 6K languages and discovered a high correlation between a word's sound and meaning both inter- and intra-cultures. aryani2020affective found that how a word sounds links to an individual's emotion. This motivates us to hypothesize that words spelled differently yet have the same meanings such as text

perturbations will also have similar sounds.

Figure 6.2 displays several perturbations that are found from real-life texts. Even though these perturbations are *spelled differently* from the original word, they all preserve *similar meanings* when perceived by humans. Such semantic preservation is feasible because humans perceive these variations *phonetically similar* to the respective original words [128]. For example, both "republican" and "republikan" sound similar when read by humans. Therefore, given the surrounding context of a perturbed sentence–e.g., "*President Trump is a* republikan", and the phonetic similarity of "republican" and "republikan", end-users are more likely to interpret the perturbed sentence as "*President Trump is a republican*". We call these characteristics of text perturbations the *SMS* property: "*similar Sound, similar Meaning, different Spellings*". Noticeably, the SMS characterization includes a subset of "visually similar" property of perturbations as studied in previous adversarial attacks such as *TextBugger* (e.g., "hello" sounds similar with "he11o"), *VIPER* and *DeepWordBug*. However, two words that look very similar sometimes carry different meanings–e.g., "garbage"→"gabrage". Moreover, our characterization is also distinguished from *homophones* (e.g., "to" and "two") which describe words with similar sound yet *different meaning*.

6.3 A Realistic Adversarial Attack

Given the above analysis, we now derive our proposed ANTHRO adversarial attack. We first share how to systematically encode the sound–i.e., phonetic feature, of any given words and use it to search for their human-written perturbations that satisfy the SMS property. Then, we introduce an iterative algorithm that utilizes the extracted perturbations to attack textual ML models.

6.3.1 Mining Perturbations in the Wild

6.3.1.1 Sound Encoding with Soundex++.

To capture the sound of a word, we adopt and extend the case-insensitive SOUNDEX algorithm. SOUNDEX helps index a word based on how it sounds rather than how it is spelled [129]. Given a word, SOUNDEX first keeps the 1st character. Then, it removes all vowels and matches the remaining characters *one by one* to a

Word	Soundex	Soundex++ (Ours)
porn	P650	P650 (**k**=0), PO650 (**k**=1)
p0rn	P065(✗)	(same as above)
lesbian	L215	L245 (**k**=0), LE245 (**k**=1)
lesbi@n	L21@(✗)	(same as above)
losbian	L215(✗)	L245 (**k**=0), LO245 (**k**=1)

(✗): Incorrect encoding

Table 6.2: SOUNDEX++ can capture visually similar characters and is more accurate in differentiating between desired (blue) and undesired (red) perturbations.

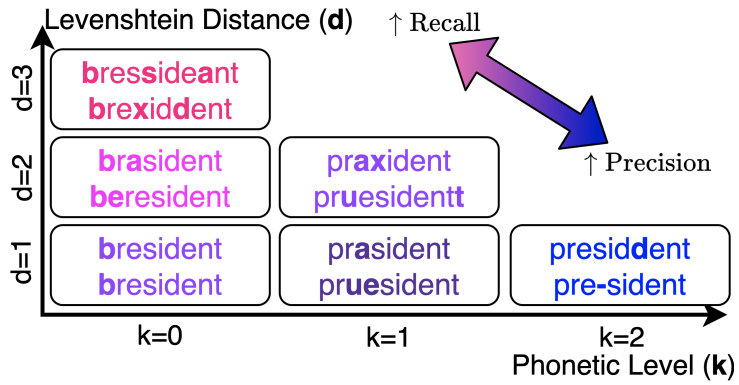


Figure 6.3: Trade-off between precision and recall of extracted perturbations for the word "president" w.r.t different **k** and **d** values. Higher **k** and lower **d** associate with better preservation of the original meaning.

digit following a set of predefined rules–e.g., "B", "F"→1, "D", "T"→3 [129]. For example, "Smith" and "Smyth" are both encoded as S530.

As the SOUNDEX system was designed mainly for encoding surnames, it does not necessarily work for texts in the wild. For example, it cannot recognize visually-similar perturbations such as "l"→"1", "a"→"@" and "O"→"0". Moreover, it always fixes the 1st character as part of the final encodes. This rule is too rigid and can result in words that are entirely different yet encoded the same (Table 6.2). To solve these issues, we propose a new SOUNDEX++ algorithm. SOUNDEX++ is equipped to both recognize visually-similar characters and encode the sound of a word at different hierarchical levels **k** (Table 6.2). Particularly, at level **k**=0, SOUNDEX++ works similar to SOUNDEX by fixing the first character. At level **k**≥1, SOUNDEX++ instead fixes the first **k**+1 characters and encodes the rest.

Dataset	#Texts	#Tokens
List of Bad Words	1.9K	1.9K
Rumours (Twitter) [131]	99K	159K
Hate Memes (Twitter) [132]	150K	328K
Personal Atks (Wiki.) [133]	116K	454K
Toxic Comments (Wiki.) (Kaggle, 2019)	2M	1.6M
Malignant Texts (Reddit) (Kaggle, 2021)	313K	857K
Hateful Comments (Reddit) (Kaggle, 2021)	1.7M	1M
Sensitive Query (Search Engine, Private)	1.2M	314K
Hateful Comments (Online News, Private)	12.7M	7M
Total texts used to extract ANTHRO	**18.3M**	-

Table 6.3: Real-life datasets that are used to extract adversarial texts in the wild, number of total examples (#Texts) and unique tokens (#Tokens) (case-insensitive)

Key	TH000	DE5263	AR000	DI630	NO300
Value (Set)	the	democrats demokRATs	are arre	dirty dirrrty	not

ANTHRO(democrats,**k**=1,**d**=1)→{democrats, demokRATs}
ANTHRO(dirty,**k**=1,**d**=2)→{dirty, dirrrty}

Table 6.4: Examples of hash table $H_1(k{=}1)$ curated from sentences *"the demokRATs are dirrrty"* and *"the democrats arre not dirty"* and its utilization.

6.3.1.2 Levenshtein Distance d and Phonetic Level k as a Semantic Preservation Proxy.

Since SOUNDEX++ is not designed to capture a word's semantic meaning, we utilize both phonetic parameter **k** and *Levenshtein distance* **d** [130] as a heuristic approximation to measure the semantic preservation between two words. Intuitively, the higher the phonetic level (**k**≥1) at which two words share the same SOUNDEX++ code and the smaller the Levenshtein distance **d** to transform one word to another, the more likely human associate them with the meaning. In other words, **k** and **d** are hyper-parameters that help control the trade-off between precision and recall when retrieving perturbations of a given word such that they satisfy the SMS property (Figure 6.3). We will later carry out a human study to evaluate how well our extracted perturbations can preserve the semantic meanings in practice.

6.3.1.3 Mining from the Wild.

To mine all human-written perturbations, we first collect a large corpus $\mathcal{D}$ of over 18M sentences written by netizens from 9 different datasets (Table 6.3). We select these datasets because they include offensive texts such as hate speech, sensitive search queries, etc., and hence very likely to include text perturbations. Next, for each phonetic level $\mathbf{k}{\leq}K$, we curate different hash tables $\{H\}_0^K$ that maps a unique SOUNDEX++ code $\mathbf{c}$ to a set of its matching unique *case-sensitive* tokens that share the same encoding $\mathbf{c}$ as follows:

$$H_{\mathbf{k}} : \mathbf{c} \mapsto \{w_j | S(w_i, k) = S(w_j, k) = \mathbf{c} \quad \forall w_i, w_j \in \mathcal{D}, w_i \neq w_j\}, \tag{6.1}$$

where $S(w, \mathbf{k})$ returns the SOUNDEX++ code of token w at phonetic level $\mathbf{k}$, K is the largest phonetic level we want to encode. With $\{H\}_0^K$, $\mathbf{k}$ and $\mathbf{d}$, we can now search for the set of perturbations $G_{\mathbf{k}}^{\mathbf{d}}(w^*)$ of a specific target token w^* as follows:

$$G_{\mathbf{k}}^{\mathbf{d}}(w^*) \leftarrow \{w_j | w_j \in H_{\mathbf{k}}[S(w^*, k)], \text{Lev}(w^*, w_j) \leq \mathbf{d}\} \tag{6.2}$$

where $\text{Lev}(w^*, w_j)$ returns the Levenshtein distance between w^* and w^j. Noticeably, we only extract $\{H\}_0^K$ **once** from $\mathcal{D}$ via Eq. (6.1), then we can use Eq. (6.2) to retrieve all perturbations for a given word during deployment. We name this method of mining and retrieving human-written text perturbations in the wild as **ANTHRO**, aka *human-like* perturbations:

$$\text{ANTHRO} : w*, \mathbf{k}, \mathbf{d}, \{H\}_0^K \quad \longmapsto \quad G_{\mathbf{k}}^{\mathbf{d}}(w^*) \tag{6.3}$$

6.3.1.4 ANTHRO Attack.

To utilize ANTHRO for adversarial attack on model $f(x)$, we propose the ANTHRO attack algorithm (Alg. 6). We use the same iterative mechanism (Ln.9–13) that is common among other black-box attacks. This process replaces the most vulnerable word in sentence x, which is evaluated with the support of Score)$(\cdot)$ function (Ln. 5), with the perturbation that best drops the prediction probability $f(x)$ on the correct label. Unlike the other methods, ANTHRO inclusively draws

Algorithm 6 ANTHRO Attack Algorithm

Input: $\{H\}_0^K$, $\mathbf{k}$, $\mathbf{d}$
Input: target classifier f, original sentence x
Output: perturbed sentence x^*
Initialize: $x^* \leftarrow x$
for word x_i **in** x **do**: $s_i \leftarrow \text{Score}(x_i, f)$
$\mathcal{W}_{\text{order}} \leftarrow \text{Sort}(x_1, x_2, ..x_m)$ according to s_i
for x_i in $\mathcal{W}_{\text{order}}$ **do**:
  $\mathcal{P} \leftarrow \text{ANTHRO}(x_i, \mathbf{k}, \mathbf{d}, \{H\}_0^K)$ // Eq.(6.3)
  $x^* \leftarrow$ replace $x_i \in x$ with the best $w \in \mathcal{P}$
  if $f(x^*) \neq f(x)$ **then** return x^*
return None

Dataset	#Total	BERT	RoBERTa
CB [133]	449K	0.84	0.84
TC (Kaggle, 2018)	160K	0.85	0.85
HS [134]	25K	0.91	0.97

Table 6.5: Evaluation datasets Cyberbullying (CB), Toxic Comments (TC) and Hate Speech (HS) and prediction performance in F1 score on their test sets of BERT and RoBERTa.

from perturbations extracted from human-written texts captured in $\{\mathcal{H}\}_0^K$ (Ln. 10). We adopt the Score(·) from *TextBugger*.

6.4 Evaluation

We evaluate ANTHRO by: (1) attack performance, (2) semantic preservation, and (3) human-likeness–i.e., how likely an attack message is spotted as machine-generated by human examiners.

6.4.1 Attack Performance

6.4.1.1 Setup.

We use BERT (*case-insensitive*) [**?**] and RoBERTa (*case-sensitive*) [135] as target classifiers to attack. We evaluate on three public tasks, namely detecting toxic comments ((TC) dataset, Kaggle 2018), hate speech ((HS) dataset [134]), and online cyberbullying texts ((CB) dataset [133]). We split each dataset to *train*, *validation*

and *test* set with the 8:1:1 ratio. Then, we use the train set to fine-tune BERT and RoBERTa with a maximum of 3 epochs and select the best checkpoint using the validation set. BERT and RoBERTa achieve around 0.85–0.97 in F1 score on the test sets (Table 6.5). We evaluate with targeted attack (change positive→negative label) since it is more practical. We randomly sample 200 examples from each test set and use them as initial sentences to attack. We repeat the process 3 times with unique random seeds and report the results. We use the *attack success rate (Atk%)* metric–i.e., the number of examples whose labels are flipped by an attacker over the total number of texts that are correctly predicted pre-attack. We use the 3rd party open-source *OpenAttack* [136] framework to run all evaluations.

6.4.1.2 Baselines.

We compare ANTHRO with three baselines, namely *TextBugger* [71], *VIPER* [122] and *DeepWordBug* [96]. These attackers utilize different character-based manipulations to craft their adversarial texts as described in Sec. 6.1. From the analysis in Sec. 6.3.1 and Figure 6.3, we set $\mathbf{k}\leftarrow 1$ and $\mathbf{d}\leftarrow 1$ for ANTHRO to achieve a balanced trade-off between precision and recall on the SMS property. We examine all attackers under several combinations of different normalization layers. They are (1) *Accents normalization* (A) and (2) *Homoglyph normalization* (H), which converts non-English accents and homoglyphs to their corresponding ascii characters, (3) *Perturbation normalization* (P), which normalizes potential character-based perturbations using the SOTA misspelling correction model *Neuspell* [123]. These normalizers are selected as counteracts against the perturbation strategies employed by *VIPER* (uses non-English accents), *DeepWordBug* (uses homoglyphs) and *TextBugger*, ANTHRO (based on misspelling and typos), respectively.

https://github.com/codebox/homoglyph

Attacker	Normalizer	BERT (*case-insensitive*)			RoBERTa (*case-sensitive*)		
		Toxic Comments	**HateSpeech**	**Cyberbullying**	**Toxic Comments**	**HateSpeech**	**Cyberbullying**
TextBugger	-	**0.76±0.02**	**0.94±0.01**	**0.78±0.03**	0.77±0.06	0.87±0.01	0.72±0.01
DeepWordBug	-	0.56±0.04	0.68±0.01	0.50±0.02	0.52±0.01	0.42±0.04	0.38±0.04
VIPER	-	0.08±0.03	0.01±0.01	0.13±0.02	**1.00±0.00**	**1.00±0.00**	**0.99±0.01**
ANTHRO	-	0.72±0.02	0.82±0.01	0.71±0.02	0.84±0.00	0.93±0.01	0.78±0.01
TextBugger	A	-	-	-	0.72±0.02	0.92±0.00	0.74±0.02
DeepWordBug	A	-	-	-	0.43±0.02	0.59±0.03	0.43±0.01
VIPER	A	-	-	-	0.09±0.01	0.05±0.01	0.17±0.02
ANTHRO	A	-	-	-	**0.77±0.02**	**0.94±0.02**	**0.84±0.02**
TextBugger	A+H	**0.78±0.03**	**0.85±0.00**	**0.79±0.00**	0.74±0.02	0.93±0.01	0.77±0.03
DeepWordBug	A+H	0.04±0.00	0.06±0.02	0.01±0.01	0.03±0.01	0.01±0.01	0.06±0.02
VIPER	A+H	0.07±0.00	0.01±0.01	0.10±0.00	0.13±0.02	0.07±0.01	0.17±0.01
ANTHRO	A+H	0.76±0.02	0.77±0.03	0.73±0.05	**0.82±0.02**	**0.97±0.00**	**0.82±0.02**
TextBugger	A+H+P	**0.73±0.02**	**0.64±0.06**	**0.70±0.04**	0.68±0.06	0.57±0.03	0.66±0.04
DeepWordBug	A+H+P	0.02±0.01	0.04±0.02	0.01±0.01	0.02±0.01	0.01±0.01	0.02±0.01
VIPER	A+H+P	0.12±0.01	0.04±0.01	0.17±0.03	0.11±0.02	0.05±0.01	0.18±0.01
ANTHRO	A+H+P	0.65±0.04	**0.64±0.01**	0.60±0.05	**0.80±0.02**	**0.91±0.03**	**0.82±0.02**

(-) BERT already has the accents normalization (A normalizer) by default, (Red): Poor performance (Atk%<0.15)

Table 6.6: Averaged attack success rate (Atk%↑) of different attack methods

6.4.1.3 Results.

Overall, both ANTHRO and *TextBugger* perform the best. Being case-sensitive, ANTHRO performs significantly better on RoBERTa and is competitive on BERT when compared to *TextBugger* (Table 6.6). This makes ANTHRO more practical since many of popular commercial APIs such as the *Perspective API* are case-sensitive–i.e., "democrats"≠"democRATs". *VIPER* achieves a near perfect score on RoBERTa, yet it is ineffective on BERT because RoBERTa uses the accent Ġ as a part of its byte-level BPE encoding [135] while BERT by default removes all such accents. Since *VIPER* exclusively utilizes accents, its attacks can be easily corrected by the *accents normalizer* (Table 6.6). Similarly, *DeepWordBug* perturbs texts with homoglyph characters, most of which can also be normalized using a 3rd party homoglyph detector (Table 6.6).

In contrast, even under all normalizers–i.e., A+H+P, *TextBugger* and ANTHRO still achieves 66.3% and 73.7% in Atk% on average across all evaluations. Although *Neuspell* [123] drops *TextBugger*'s Atk% 14.7% across all runs, it can only reduce the Atk% of ANTHRO a mere 7.5% on average. This is because *TextBugger* and *Neuspell* or other dictionary-based typo correctors rely on fixed deductive rules–e.g., swapped, replaced by neighbor letters, for attack and defense. However, ANTHRO utilizes human-written perturbations which are greatly varied, hence less likely to be systematically detected. We further discuss the limitation of misspelling correctors such as NeuSpell in Sec. 6.7.

6.4.2 Human Evaluation

Since ANTHRO and *TextBugger* are the top two effective attacks, this section will focus on evaluating their ability in semantic preservation and human-likeness. Given an original sentence x and its adversarial text x^* generated by either one of the attacks, we design a human study to *directly compare* ANTHRO with *TextBugger*. Specifically, two alternative hypotheses for our validation are (1) $\mathcal{H}_{\text{Semantics}}$: x^* generated by ANTHRO preserves the original meanings of x *better* than that generated by *TextBugger* and (2) $\mathcal{H}_{\text{Human}}$: x^* generated by ANTHRO is *more likely* to be perceived as a human-written text (and not machine) than that generated by *TextBugger*.

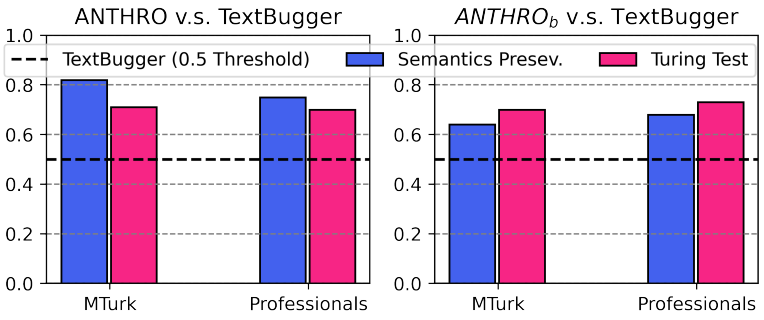


Figure 6.4: Semantic preservation and human-likeness

6.4.2.1 Human Study Design.

We use the two attackers to generate adversarial texts targeting BERT model on 200 examples sampled from the TC dataset's test set. We then gather examples that are successfully attacked by both ANTHRO and *TextBugger*. Next, we present a pair of texts, one generated by ANTHRO and one by *TextBugger*, together with the original sentence to human subjects. We then ask them to select (1) which text *better* preserves the meaning of the original sentence and (2) which text is *more likely* to be written by human. To reduce noise and bias, we also provide a *"Cannot decide"* option when quality of both texts are equally good or bad, and present the two questions in two separate tasks. Since the definition of semantic preservation can be subjective, we recruit human subjects as both (1) Amazon Mechanical Turk (MTurk) workers and (2) professional data annotators at a company with extended experience in annotating texts in domain such as toxic and hate speech.

We recruit MTurk workers who are 18 years or older residing in North America. MTurk workers are recruited using the following qualifications provided by AMT, namely (1) recognized as "master" workers by AMT system, (2) have done at least 5K HITs and (3) have historical HITs approval rate of at least 98%. These qualifications are also more conservative than previous human studies we found in previous literature. We pay each worker on average around $10 an hour or higher (federal minimum wage was $7.25 in 2021 when we carried out our study). To limit abusive behaviors, we impose a minimum attention span of 30 seconds for the workers to complete each task. Our human subject study with MTurk workers was IRB-approved.

Attacker	Normalizer	**BERT** (*case-insensitive*)			**RoBERTa** (*case-sensitive)*		
		Toxic Comments	**HateSpeech**	**Cyberbullying**	**Toxic Comments**	**HateSpeech**	**Cyberbullying**
TextBugger	-	0.76±0.02	0.94±0.01	0.78±0.03	0.77±0.06	0.87±0.01	0.72±0.01
ANTHRO$_\beta$	-	**0.82±0.01**	**0.97±0.01**	**0.88±0.04**	**0.91±0.02**	**0.97±0.01**	**0.89±0.02**
TextBugger	A+H+P	0.73±0.02	0.64±0.06	0.70±0.04	0.68±0.06	0.57±0.03	0.66±0.04
ANTHRO$_\beta$	A+H+P	**0.85±0.04**	**0.79±0.02**	**0.84±0.03**	**0.88±0.04**	**0.93±0.01**	**0.91±0.01**

Table 6.7: Averaged attack success rate (Atk%↑) of ANTHRO$_\beta$ and *TextBugger*

Reason	Favorable From GRACE	Unfavorable From TextBugger
Genuine Typos	stuupid, but, Faoggt	sutpid, burt, Foggat
Intelligible	faiilure	faioure
Sound Preserv.	shytty, crp	shtty, crsp
Meaning Preserv.	ga-y, ashole, dummb	bay, alshose, dub
High Search Results	sodmized, kiills	Smdooized, klils
Personal Exposure	ign0rant, gaarbage	ignorajt, garage
Word Selection	morons→mor0ns	edited→ewited

Table 6.8: Top reasons in favoring ANTHRO's perturbations as more likely to be written by human.

6.4.2.2 Quantitative Results.

Experiments show that it is statistically significant (p-*value*$\leq$0.05) to reject the null hypotheses of both $\mathcal{H}_{\text{Semantics}}$ and $\mathcal{H}_{\text{Human}}$. Overall, adversarial texts generated by perturbations mined in the wild are much better at preserving the original semantics and also at resembling human-written texts than those generated by *TextBugger* (Figure 6.4, Left).

6.4.2.3 Qualitative Analysis.

Table 6.8 summarizes the top reasons why they favor ANTHRO over *TextBugger* in terms of human-likeness. ANTHRO's perturbations are perceived similar to genuine typos and more intelligible. They also better preserve both meanings and sounds. Moreover, some annotators also rely on personal exposure on Reddit, YouTube comments, or the frequency of word use via the search function on Reddit to decide if a word-choice is human-written.

6.5 ANTHRO$_\beta$ Attack

We examine if perturbations inductively extracted from the wild help improve the deductive *TextBugger* attack. Hence, we introduce ANTHRO$_\beta$, which considers the perturbation candidates from both ANTHRO and *TextBugger* in Ln. 10 of Alg. 6. Alg. 6 still selects the perturbation that best flip the target model's prediction.

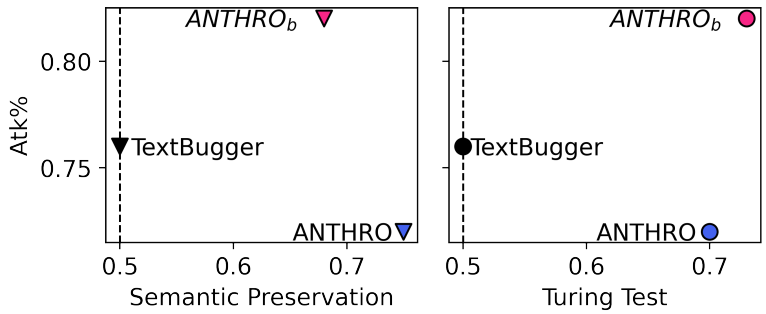


Figure 6.5: Trade-off among evaluation metrics

6.5.0.1 Attack Performance.

Even though ANTHRO comes second after *TextBugger* when attacking BERT model, Table 6.7 shows that when combined with *TextBugger*–i.e., ANTHRO$_\beta$, it consistently achieves superior performance with an average of 82.7% and 90.7% in Atk% on BERT and RoBERTa even under all normalizers (A+H+P).

6.5.0.2 Semantic Preservation and Human-Likeness.

ANTHRO$_\beta$ improves *TextBugger*'s Atk%, semantic preservation and human-likeness score with an increase of over 8%, 32% and 42% (from 0.5 threshold) on average (Table 6.7, 6.4, Right), respectively. The presence of only a few human-like perturbations generated by ANTHRO is sufficient to signal whether or not the whole sentence is written by humans, while only one unreasonable perturbation generated by *TextBugger* can adversely affect its meaning. This explains the performance drop in terms of semantic preservation but not in human-likeness when indirectly comparing ANTHRO$_\beta$ with ANTHRO. Overall, ANTHRO$_\beta$ also has the best trade-off between Atk% and human evaluation–i.e., positioning at top right corners in Figure 6.5, with a noticeable superior Atk%.

6.6 Defend ANTHRO, ANTHRO$_\beta$ Attack

6.6.1 Proposed Defense

We suggest two countermeasures against ANTHRO attack. They are **(i) Sound-Invariant Model (SoundCNN):** When the defender do *not* have access to $\{\mathcal{H}\}_0^K$

Model	ANTHRO			ANTHRO$_\beta$		
	TC↓	HS↓	CB↓	TC↓	HS↓	CB↓
BERT	0.72	0.82	0.71	0.82	0.97	0.88
BERT+A+H+P	0.65	0.65	0.60	0.85	0.79	0.84
ADV.TRAIN	0.41	0.30	0.35	**0.72**	**0.72**	**0.67**
SOUNDCNN	**0.14**	**0.02**	**0.15**	0.86	0.84	0.92

Table 6.9: Averaged Atk%↓ of ANTHRO and ANTHRO$_\beta$ against different defense models.

learned by the attacker, the defender trains a generic model that encodes not the spellings but the phonetic features of a text for prediction. Here we train a CNN model [137] on top of a embeddings layer for discrete SOUNDEX++ encodings of each token in a sentence; **(ii) Adversarial Training (Adv.Train):** To overcome the lack of access to $\{\mathcal{H}\}_0^K$, the defender extracts his/her perturbations in the wild from a separate corpus $\mathcal{D}^*$ where $\mathcal{D}^* \cap \mathcal{D} = \emptyset$ and uses them to augment the training examples–i.e., via self-attack with ratio 1:1, to fine-tune a more robust BERT model. We use $\mathcal{D}^*$ as a corpus of 34M general comments from online news.

6.6.2 Results.

We compare the two defenses against BERT and BERT combined with 3 layers of normalization A+H+P. BERT is selected as it is better than RoBERTa at defending against ANTHRO (Table 6.6). Table 6.9 shows that both SOUNDCNN and ADV.TRAIN are robust against ANTHRO attack, while ADV.TRAIN performs best when defending ANTHRO$_\beta$. Since SOUNDCNN is strictly based on phonetic features, it is vulnerable against ANTHRO$_\beta$ whenever *TextBugger*'s perturbations are selected. Table 6.9 also underscores that ANTHRO$_\beta$ is a strong and practical attack, defense against which is thus an important future direction.

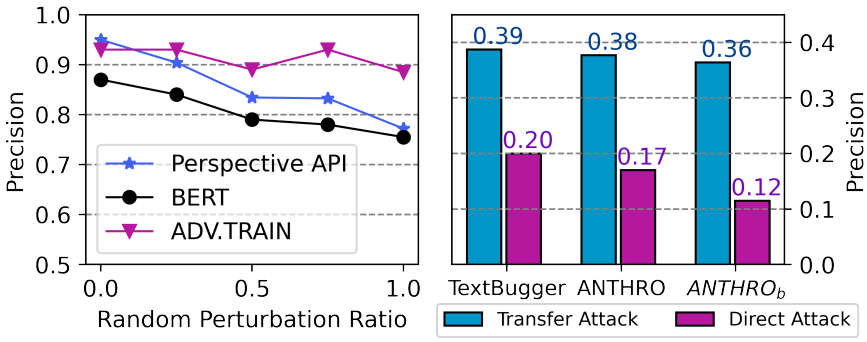


Figure 6.6: (Left) Precision on human-written perturbed texts synthesized by ANTHRO and (Right) Robustness evaluation of *Perspective API* under different attacks

6.7 Discussion and Analysis

6.7.1 Evaluation with *Perspective API*.

We evaluate if ANTHRO and ANTHRO$_\beta$ can successfully attack the popular *Perspective API* , which has been adopted in various publishers–e.g., NYTimes, and platforms–e.g., Disqus, Reddit, to detect toxicity. We evaluate on 200 toxic texts randomly sampled from the TC dataset. Figure 6.6 (Left) shows that the API provides superior performance compared to a self fine-tuned BERT classifier, yet its precision deteriorates quickly from 0.95 to only 0.9 and 0.82 when 25%–50% of a sentence are randomly perturbed using human-written perturbations. However, the Adv.Train (Sec. 6.6) model achieves fairly consistent precision in the same setting. This shows that ANTHRO is not only a powerful and realistic attack, but also can help develop more robust text classifiers in practice. The API is also vulnerable against both direct (Alg. 6) and transfer ANTHRO attacks through an intermediate BERT classifier, with its precision dropped to only 0.12 when evaluated against ANTHRO$_\beta$.

6.7.2 Generalization beyond Offensive Texts.

Although ANTHRO extracts perturbations from abusive data, the majority of them are non-abusive texts. Thus, ANTHRO learns perturbations for non-abusive

`https://www.perspectiveapi.com/`

Task	Sentiment Analysis	Categorization
ANTHRO	0.80	0.93
ANTHRO$_\beta$	0.86	1.00

Table 6.10: Attack success rate (Atk%↑) of ANTHRO and ANTHRO$_\beta$ on non-abusive task domains.)

English words–e.g., hilarious->Hi-Larious, shot->sh • t. We also make no assumption on the task domains that ANTHRO can attack. Evidently, ANTHRO and ANTHRO$_\beta$ achieves 80%, 86% Atk% and 90%, 100% Atk% on fooling Google's sentiment analysis API (untargeted attack using 200 randomly selected texts of the SST dataset [138]) and text categorization API (untargeted attack on 50 randomly selected news with original label of "SPORT" from the BBC News dataset (Table 6.10).

6.7.3 Limitation of Misspelling Correctors.

Similar to other spell-checkers such as *pyspellchecker* and *symspell*, the SOTA NeuSpell depends on a fixed dictionary of common misspellings, or synthetic misspellings generated by random permutation of characters [123]. These checkers often assume perturbations are within an edit-distance threshold from the original words. This makes them exclusive since one can easily generate new perturbations by repeating a specific character–e.g., "porn"→"pooorn". Also, due to the iterative attack mechanism (Alg. 6) where each token in a sentence is replaced by many candidates until the correct label's prediction probability drops, ANTHRO only needs a single good perturbation that is not detected by NeuSpell for a successful replacement. Thus, by formulating perturbations by not only their spellings but also their sounds, ANTHRO is able to mine perturbations that can circumvent NeuSpell.

6.7.4 Computational Analysis

The **one-time** extraction of $\{\mathcal{H}\}_0^K$ via Eq. (6.1) has $\mathcal{O}(|\mathcal{D}|L)$ where $|\mathcal{D}|$, L is the # of tokens and the length of longest token in $\mathcal{D}$ (hash-map operations cost $\mathcal{O}(1)$).

`https://cloud.google.com/natural-language`
`https://www.kaggle.com/c/learn-ai-bbc/`

Given a word w and $\mathbf{k}, \mathbf{d}$, GRACE retrieves a list of perturbation candidates via Eq. (6.2) with $\mathcal{O}(|w|max(\mathcal{H}_k))$ where $|w|$ is the length of w and $max(\mathcal{H}_k)$ is the size of the largest set of tokens sharing the same SOUNDEX++ encoding in $\mathcal{H}_k$. Since $max(\mathcal{H}_k)$ is constant, the upper-bound then becomes $\mathcal{O}(|w|)$.

6.7.5 Limitation

The perturbation candidate retrieval operation (Eq. (6.2)) has a higher computational complexity than that of other methods–i.e., $\mathcal{O}(|w|)$ v.s. $\mathcal{O}(1)$ where $|w|$ is the length of an input token w. This can prolong the running time, especially when attacking long documents. However, we can overcome this by storing all the perturbations (given $\mathbf{k}, \mathbf{d}$) of the top frequently used offensive and non-offensive English words. We can then expect the operation to have an average complexity close to $\mathcal{O}(1)$. The current SOUNDEX++ algorithm is designed for English texts and might not be applicable in other languages. Thus, we plan to extend ANTHRO to a multilingual setting.

6.8 Conclusion

We propose ANTHRO, a character-based attack algorithm that extracts human-written perturbations in the wild and then utilizes them for adversarial text generation. Our approach yields the best trade-off between attack performance, semantic preservation and stealthiness under both empirical experiments and human studies. A BERT classifier trained with examples augmented by ANTHRO can also better understand human-written texts.

Chapter 7
Conclusion

This thesis introduces the concept of *trustworthy machine learning* as ML models that do not only make accurate predictions on unseen examples but are also robust under several practical constraints such as effective learning under limited labeled data, transparent and explainable to the end-users, and robust against adversarial attacks. Specifically, this thesis proposes to use VAE-based generative models to synthesize clickbaits as additional positive examples to train better clickbait, detection models. Moreover, it introduces GRACE, a novel algorithm that enables any neural network models to effectively generate useful explanation texts for their predictions. Furthermore, the thesis also proposes MALCOM, a neural network-based conditional text generator that can synthesize high-quality malicious comments to attack a wide range of white-box and black-box fake news detectors. Furthermore, this thesis explores the use of "honeypot", a concept from cybersecurity, to bait and trap potential malicious universal trigger attacks in the NLP domain. Last but not least, this thesis proposes an *inductive approach* to mine text perturbations in the wild [7]. The extracted human-written perturbations are then utilized to derive a more realistic textual attack and defense. Although all of the technical contributions of this thesis specifically focus on various classification tasks in the text domain, their motivation and intuition are also applicable in other NLP tasks–e.g., natural language generation, and domains–e.g., computer vision. Especially, this thesis also lays a foundation for several later works in the area of security ML such as *attack-invariant adversarial defense* [139], *multi-goals adversarial attacks* [140]. Through these technical contributions, this thesis also hopes to contribute to the adoption of ML systems in high-stakes fields where mutual trust between humans and machines is paramount.

Bibliography

[1] Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2002) "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, **16**, pp. 321–357.

[2] Le, T., K. Shu, M. D. Molina, D. Lee, S. S. Sundar, and H. Liu "5 Sources of Clickbaits You Should Know! Using Synthetic Clickbaits to Improve Prediction and Distinguish between Bot-Generated and Human-Written Headlines," *IEEE/ACM ASONAM'19*.

[3] Le, T., S. Wang, and D. Lee "GRACE: Generating Concise and Informative Contrastive Sample to Explain Neural Network Model's Prediction," *KDD'20*.

[4] Ribeiro, M. T., S. Singh, and C. Guestrin (2016) ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD/KDD, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144.

[5] Le, T., S. Wang, and D. Lee "MALCOM: Generating Malicious Comments to Attack Neural Fake News Detection Models," in *ICDM'20*.

[6] Le, T., N. Park, and D. Lee "A Sweet Rabbit Hole by DARCY: Using Honeypots to Detect Universal Trigger's Adversarial Attacks," in *ACL'21*.

[7] Le, T., J. Lee, K. Yen, Y. Hu, and D. Lee "Perturbations in the Wild: Leveraging Human-Written Text Perturbations for Realistic Adversarial Attack and Defense," *ACL'22 (Findings)*.

[8] Rony, M. M. U., N. Hassan, and M. Yousuf (2017) "Diving Deep into Clickbaits: Who Use Them to What Extents in Which Topics with What Effects?" in *IEEE/ACM ASONAM 2017*, pp. 232–239.

[9] Chen, Y., N. J. Conroy, and V. L. Rubin (2015) "Misleading Online Content: Recognizing Clickbait As "False News"," in *ACM WMDD 2015*, New York, NY, USA, pp. 15–19.

[10] Cao, X., T. Le, Jason, and Zhang (2017) "Machine Learning Based Detection of Clickbait Posts in Social Media," *arXiv preprint arXiv:1710.01977*, 1710.01977.

[11] Couldry, N. and J. Turow (2014) "Big Data, Big Questions| Advertising, Big Data and the Clearance of the Public Realm: Marketers' New Approaches to the Content Subsidy," *International Journal of Communication*, **8**(0), pp. 1710–1726.

[12] Potthast, M., S. Köpsel, B. Stein, and M. Hagen (2016) "Clickbait detection," in *European Conference on Information Retrieval*, Springer, pp. 810–817.

[13] Wei, W. and X. Wan (2017) "Learning to Identify Ambiguous and Misleading News Headlines," in *IJCAI 2017*, AAAI Press, pp. 4172–4178.

[14] Chakraborty, A., B. Paranjape, S. Kakarla, and N. Ganguly (2016) "Stop clickbait: Detecting and preventing clickbaits in online news media," in *IEEE/ACM ASONAM 2016*, pp. 9–16.

[15] Bowman, S. R., L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio (2015) "Generating Sentences from a Continuous Space," *CoRR*, **abs/1511.06349**.

[16] Biyani, P., K. Tsioutsiouliklis, and J. Blackmer (2016) "" 8 Amazing Secrets for Getting More Clicks": Detecting Clickbaits in News Streams Using Article Informality." in *AAAI*, pp. 94–100.

[17] Zhao, S., J. Song, and S. Ermon (2017) "InfoVAE: Information Maximizing Variational Autoencoders," *arXiv preprint arXiv:1706.02262*, 1706.02262.

[18] Jeni, L. A., J. F. Cohn, and F. De La Torre (2013) "Facing imbalanced data–recommendations for the use of performance metrics," in *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, IEEE, pp. 245–251.

[19] Tenenbaum, J. B., V. de Silva, and J. C. Langford (2000) "A global geometric framework for non linear dimensionality reduction," *Science*, **290**(5500), pp. 2319–2323.

[20] Miao, Y. and P. Blunsom (2016) "Language as a Latent Variable: Discrete Generative Models for Sentence Compression," in *EMNLP 2016*, pp. 319–328.

[21] Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017) "Controllable text generation," *arXiv preprint arXiv:1703.00955*.

[22] Abokhodair, N., D. Yoo, and D. W. McDonald (2015) "Dissecting a Social Botnet: Growth, Content and Influence in Twitter," in *ACM CSCW 2015*, New York, NY, USA, pp. 839–851.

[23] Geer, J. G. and K. F. Kahn (1993) "Grabbing attention: An experimental investigation of headlines during campaigns," *Political Communication*, **10**(2), pp. 175–191.

[24] Dor, D. (2003) "On newspaper headlines as relevance optimizers," *Journal of Pragmatics*, **35**(5), pp. 695–721.

[25] Agrawal, A. (2016) "Clickbait detection using deep learning," in *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, IEEE, pp. 268–272.

[26] Miao, Y., L. Yu, and P. Blunsom (2016) "Neural variational inference for text processing," in *ICML 2016*, pp. 1727–1736.

[27] Elyashar, A., J. Bendahan, and R. Puzis (2017) "Detecting Clickbait in Online Social Media: You Won't Believe How We Did It," *arXiv preprint arXiv:1710.06699*.

[28] Gairola, S., Y. K. Lal, V. Kumar, and D. Khattar (2017) "A Neural Clickbait Detection Engine," *arXiv preprint arXiv:1710.01507*.

[29] Shavitt, I. and E. Segal (2018) "Regularization learning networks: deep learning for tabular datasets," in *2018 NIPS*, pp. 1379–1389.

[30] Arik, S. O. and T. Pfister (2019) "TabNet: Attentive Interpretable Tabular Learning," *arXiv preprint arXiv:1908.07442*.

[31] Barz, B. and J. Denzler (2019) "Deep Learning on Small Datasets without Pre-Training using Cosine Loss," *arXiv preprint arXiv:1901.09054*.

[32] Marais, J. A. (2019) *Deep learning for tabular data: an exploratory study*, Ph.D. thesis, Stellenbosch: Stellenbosch University.

[33] Ravì, D., C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G.-Z. Yang (2016) "Deep learning for health informatics," *IEEE journal of biomedical and health informatics*, **21**(1), pp. 4–21.

[34] Bejnordi, B. E., M. Veta, P. J. Van Diest, B. Van Ginneken, N. Karssemeijer, G. Litjens, J. A. Van Der Laak, M. Hermsen, Q. F. Manson, M. Balkenhol, et al. (2017) "Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer," *Jama*, **318**(22), pp. 2199–2210.

[35] Fischer, T. and C. Krauss (2018) "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, **270**(2), pp. 654–669.

[36] Dixon, M. F., N. G. Polson, and V. O. Sokolov (2019) "Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading," *Applied Stochastic Models in Business and Industry*, **35**(3), pp. 788–807.

[37] Kosinski, M., D. Stillwell, and T. Graepel (2013) "Private traits and attributes are predictable from digital records of human behavior," *Proceedings of the national academy of sciences*, **110**(15), pp. 5802–5805.

[38] Zizzo, D. J., D. Sgroi, et al. (2000) *Bounded-rational behavior by neural networks in normal form games*, Nuffield College.

[39] Mahdavifar, S. and A. A. Ghorbani (2019) "Application of deep learning to cybersecurity: A survey," *Neurocomputing*, **347**, pp. 149–176.

[40] Xu, X., C. Liu, Q. Feng, H. Yin, L. Song, and D. Song (2017) "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proceedings of the 2017 ACM SIGSAC CCS*, pp. 363–376.

[41] Zeiler, M. D. and R. Fergus (2014) "Visualizing and understanding convolutional networks," in *European conference on computer vision*, Springer, pp. 818–833.

[42] Simonyan, K., A. Vedaldi, and A. Zisserman (2013) "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*.

[43] Petsiuk, V., A. Das, and K. Saenko (2018) "Rise: Randomized input sampling for explanation of black-box models," in *BMVC*.

[44] Li, J., X. Chen, E. Hovy, and D. Jurafsky (2015) "Visualizing and understanding neural models in nlp," *arXiv preprint arXiv:1506.01066*.

[45] Chu, L., X. Hu, J. Hu, L. Wang, and J. Pei (2018) "Exact and consistent interpretation for piecewise linear neural networks: A closed form solution," in *Proceedings of the 24th ACM SIGKDD/KDD*, ACM, pp. 1244–1253.

[46] Lipton, Z. C. (2018) "The mythos of model interpretability," *Queue*, **16**(3).

[47] Wu, E. and S. Madden (2013) "Scorpion: Explaining away outliers in aggregate queries," *Proceedings of the VLDB Endowment*, **6**(8), pp. 553–564.

[48] Meliou, A., S. Roy, and D. Suciu (2014) "Causality and explanations in databases," *Proceedings of the VLDB Endowment*, **7**(13), pp. 1715–1716.

[49] Roy, S. and D. Suciu (2014) "A formal approach to finding explanations for database queries," in *Proceedings of the 2014 ACM SIGMOD/PODS*, ACM, pp. 1579–1590.

[50] Silverstein, C., S. Brin, R. Motwani, and J. Ullman (2000) "Scalable techniques for mining causal structures," *Data Mining and Knowledge Discovery*, **4**(2-3), pp. 163–192.

[51] Moosavi-Dezfooli, S.-M., A. Fawzi, and P. Frossard (2016) "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the 2016 IEEE CVPR*, pp. 2574–2582.

[52] Mao, X., Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley (2017) "Least squares generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802.

[53] Flannery, B. P., S. A. Teukolsky, W. H. Press, and W. T. Vetterling (1988) *Numerical recipes in C: The art of scientific computing*, vol. 2, Cambridge University Press.

[54] Yu, L. and H. Liu (2003) "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings of the 20th ICML*, pp. 856–863.

[55] Dua, D. and C. Graff (2017), "UCI Machine Learning Repository," . URL `http://archive.ics.uci.edu/ml`

[56] Datta, A., S. Sen, and Y. Zick (2016) "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems," in *2016 IEEE symposium on security and privacy (SP)*, IEEE, pp. 598–617.

[57] Wachter, S., B. Mittelstadt, and C. Russell (2017) "Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GPDR," *Harv. JL & Tech.*, **31**, p. 841.

[58] Zhang, X., A. Solar-Lezama, and R. Singh (2018) "Interpreting neural network judgments via minimal, stable, and symbolic corrections," in *Advances in Neural Information Processing Systems*, pp. 4874–4885.

[59] van der Waa, J., M. Robeer, J. van Diggelen, M. Brinkhuis, and M. Neerincx (2018) "Contrastive explanations with local foil trees," *arXiv preprint arXiv:1806.07470*.

[60] Khanna, R., E. Elenberg, A. G. Dimakis, S. Negahban, and J. Ghosh (2017) "Scalable greedy feature selection via weak submodularity," .

[61] Hindman, M. and V. Barash (2018) "Disinformation, and Influence Campaigns on Twitter," *Knight Foundation: George Washington University*.

[62] Allen, J., B. Howland, M. Mobius, D. Rothschild, and D. J. Watts (2020) "Evaluating the fake news problem at the scale of the information ecosystem," *Science Advances*, **6**(14), p. eaay3539.

[63] Aldwairi, M. and A. Alwahedi (2018) "Detecting fake news in social media networks," *Procedia Computer Science*, **141**, pp. 215–222.

[64] Ruchansky, N., S. Seo, and Y. Liu (2017) "Csi: A hybrid deep model for fake news detection," in *CIKM'17*, ACM, pp. 797–806.

[65] Shu, K., L. Cui, S. Wang, D. Lee, and H. Liu "dEFEND: Explainable Fake News Detection," .

[66] Cui, L. and S. W. D. Lee "SAME: Sentiment-Aware Multi-Modal Embedding for Detecting Fake News," .

[67] Qian, F., C. Gong, K. Sharma, and Y. Liu (2018) "Neural User Response Generator: Fake News Detection with Collective User Intelligence." in *IJCAI'18*, pp. 3834–3840.

[68] Papernot, N., P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami (2016) "The limitations of deep learning in adversarial settings," in *EuroS&P'16*, IEEE, pp. 372–387.

[69] Zhou, Z., H. Guan, M. M. Bhat, and J. Hsu "Fake News Detection via NLP is Vulnerable to Adversarial Attacks," in *ICAART'19*, SciTePress.

[70] Horne, B. D., J. Nørregaard, and S. Adali "Robust Fake News Detection Over Time and Attack," *ACM TIST'9*.

[71] Li, J., S. Ji, T. Du, B. Li, and T. Wang (2018) "TextBugger: Generating Adversarial Text Against Real-world Applications," *NDSS'18*.

[72] Cer, D., Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al. (2018) "Universal sentence encoder," *EMNLP'18, Demo*.

[73] Ebrahimi, J., A. Rao, D. Lowd, and D. Dou "HotFlip: White-Box Adversarial Examples for Text Classification," in *ACL'18*.

[74] Pruthi, D., B. Dhingra, and Z. C. Lipton (2019) "Combating Adversarial Misspellings with Robust Word Recognition," in *ACL'19*.

[75] Wang, Z., H. Liu, J. Tang, S. Yang, G. Y. Huang, and Z. Liu (2019) "Learning Multi-level Dependencies for Robust Word Recognition," *arXiv preprint arXiv:1911.09789*.

[76] Shu, K., S. Wang, T. Le, D. Lee, and H. Liu (2018) "Deep headline generation for clickbait detection," in *ICDM'18)*, IEEE, pp. 467–476.

[77] Lamb, A. M., A. G. A. P. Goyal, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio (2016) "Professor forcing: A new algorithm for training recurrent networks," in *NIPS'16*.

[78] Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014) "Generative adversarial networks," *NIPS'14*, pp. 2672–2680.

[79] Jolicoeur-Martineau, A. (2018) "The relativistic discriminator: a key element missing from standard GAN," *ICLR'19*.

[80] Gabielkov, M., A. Ramachandran, A. Chaintreau, and A. Legout (2016) "Social Clicks: What and Who Gets Read on Twitter?" .
URL `https://hal.inria.fr/hal-01281190`

[81] Goodfellow, I., Y. Bengio, and A. Courville (2016) *Deep learning*, MIT press.

[82] Jang, E., S. Gu, and B. Poole (2016) "Categorical reparameterization with gumbel-softmax," *ICLR'17*.

[83] Nie, W., N. Narodytska, and A. Patel (2019) "RelGAN: Relational Generative Adversarial Networks for Text Generation," in *ICLR'19*.

[84] Santoro, A., R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap (2018) "Relational recurrent neural networks," in *NIPS'18*, pp. 7299–7310.

[85] Shu, K., D. Mahudeswaran, S. Wang, D. Lee, and H. Liu (2018) "FakeNewsNet: A Data Repository with News Content, Social Context and Dynamic Information for Studying Fake News on Social Media," *arXiv preprint arXiv:1809.01286*.

[86] Zhang, J., B. Dong, and P. Yu "FAKEDETECTOR: Effective fake news detection with deep diffusive neural network," in *ICDE'20*.

[87] Yu, L., W. Zhang, J. Wang, and Y. Yu (2017) "Seqgan: Sequence generative adversarial nets with policy gradient," in *AAAI'17*.

[88] Guo, B., H. Wang, Y. Ding, S. Hao, Y. Sun, and Z. Yu (2019), "c-TextGen: Conditional Text Generation for Harmonious Human-Machine Interaction," 1909.03409.

[89] Pennebaker, J. W., R. L. Boyd, K. Jordan, and K. Blackburn (2015) "The development and psychometric properties of LIWC2015," . URL https://eprints.lancs.ac.uk/id/eprint/134191

[90] Yuan, D., Y. Miao, N. Z. Gong, Z. Yang, Q. Li, D. Song, Q. Wang, and X. Liang (2019) "Detecting Fake Accounts in Online Social Networks at the Time of Registrations," in *CCS'19*, pp. 1423–1438.

[91] Goodfellow, I., J. Shlens, and C. Szegedy "Explaining and Harnessing Adversarial Examples," in *ICLR'15*.

[92] Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017) "Attention is all you need," in *NIPS'17*, pp. 5998–6008.

[93] Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017) "Toward controlled generation of text," in *ICML'17*, JMLR. org, pp. 1587–1596.

[94] Zellers, R., A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi (2019) "Defending Against Neural Fake News," *arXiv preprint arXiv:1905.12616*.

[95] Wallace, E., S. Feng, N. Kandpal, M. Gardner, and S. Singh "Universal Adversarial Triggers for Attacking and Analyzing NLP," in *EMNLP-IJCNLP'19*.

[96] Gao, J., J. Lanchantin, M. L. Soffa, and Y. Qi (2018) "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *SPW'18*, IEEE, pp. 50–56.

[97] Garg, S. and G. Ramakrishnan (2020) "BAE: BERT-based Adversarial Examples for Text Classification," *EMNLP'20*.

[98] Gan, W. C. and H. T. Ng "Improving the robustness of question answering systems to question paraphrasing," in *ACL'19*.

[99] Cheng, M., J. Yi, P.-Y. Chen, H. Zhang, and C.-J. Hsieh "Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples," in *AAAI'20*.

[100] Pang, B. and L. Lee (2005) "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales," in *ACL'05*.

[101] Wang, A., A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman (2019) "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," in *ICLR'19*.

[102] Pang, B. and L. Lee (2004) "A Sentimental Education: Sentiment Analysis Using Subjectivity," in *ACL'04*, pp. 271–278.

[103] Zhang, X., J. Zhao, and Y. LeCun "Character-level convolutional networks for text classification," in *NIPS'15*.

[104] Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL-HLT'19*.

[105] Smith, L. and Y. Gal (2018) "Understanding measures of uncertainty for adversarial example detection," *arXiv preprint arXiv:1803.08533*.

[106] Ma, X., B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, G. Schoenebeck, D. Song, M. E. Houle, and J. Bailey (2018) "Characterizing adversarial subspaces using local intrinsic dimensionality," *ICLR'18*.

[107] Papernot, N., P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami "Practical black-box attacks against machine learning," in *ASIACCS'17*.

[108] Gunning, R. et al. (1952) *Technique of clear writing*, McGraw-Hill.

[109] Wang, B., Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao (2019) "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *EuroS&P'19*, IEEE, pp. 707–723.

[110] Liu, Y., W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang (2019) "ABS: Scanning neural networks for back-doors by artificial brain stimulation," in *CCS'19*.

[111] Qiao, X., Y. Yang, and H. Li (2019) "Defending neural backdoors via generative distribution modeling," in *NIPS'19*, pp. 14004–14013.

[112] Shan, S., E. Wenger, B. Wang, B. Li, H. Zheng, and B. Y. Zhao (2019) "Using Honeypots to Catch Adversarial Attacks on Neural Networks," *CCS'20*.

[113] Chen, C. and J. Dai (2020) "Mitigating backdoor attacks in LSTM-based Text Classification Systems by Backdoor Keyword Identification," *arXiv preprint arXiv:2007.12070*.

[114] Qi, F., Y. Chen, M. Li, Z. Liu, and M. Sun (2020) "ONION: A Simple and Effective Defense Against Textual Backdoor Attacks," *arXiv preprint arXiv:2011.10369*.

[115] Liu, K., B. Dolan-Gavitt, and S. Garg (2018) "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer, pp. 273–294.

[116] Paganini, M. and J. Forde (2020) "Streamlining Tensor and Network Pruning in PyTorch," *arXiv preprint arXiv:2004.13770*.

[117] Wang, X., H. Jin, and K. He (2019) "Natural language adversarial attacks and defenses in word level," *arXiv preprint arXiv:1909.06723*.

[118] Sheikholeslami, F., S. Jain, and G. B. Giannakis (2020) "Minimum uncertainty based detection of adversaries in deep neural networks," in *2020 Information Theory and Applications Workshop (ITA)*, IEEE, pp. 1–16.

[119] Pang, T., C. Du, Y. Dong, and J. Zhu (2018) "Towards robust detection of adversarial examples," in *NIPS'18*, pp. 4579–4589.

[120] Carlini, N. (2020) "A Partial Break of the Honeypots Defense to Catch Adversarial Attacks," *arXiv preprint arXiv:2009.10975*.

[121] Wang, W., L. Wang, R. Wang, Z. Wang, and A. Ye (2019) "Towards a Robust Deep Neural Network in Texts: A Survey," *arXiv preprint arXiv:1902.07285*.

[122] Eger, S., G. G. Şahin, A. Rücklé, J.-U. Lee, C. Schulz, M. Mesgar, K. Swarnkar, E. Simpson, and I. Gurevych "Text Processing Like Humans Do: Visually Attacking and Shielding NLP Systems," in *NAACL'19*.

[123] Jayanthi, S. M., D. Pruthi, and G. Neubig "NeuSpell: A Neural Spelling Correction Toolkit," in *EMNLP'20, Demo*.

[124] Tagg, C. (2011) "Wot did he say 01" could u not c him 4 dust?: Written and Spoken Creativity in Text Messaging," *Transforming literacies and language: Multimodality and literacy in the new media age*, **223**.

[125] Köhler, W. (1967) "Gestalt psychology," *Psychologische Forschung*, **31**(1), pp. XVIII–XXX.

[126] Jared, D. and M. S. Seidenberg (1991) "Does word identification proceed from spelling to sound to meaning?" *Journal of Experimental Psychology: General*, **120**(4), p. 358.

[127] Gough, P. B., J. Kavanagh, and I. Mattingly (1972) "One second of reading," *Cambridge: MIT Press*, pp. 331–358.

[128] Van Orden, G. C. (1987) "A ROWS is a ROSE: Spelling, sound, and reading," *Memory & cognition*, **15**(3), pp. 181–198.

[129] Stephenson, C. (1980) "The methodology of historical census record linkage: A user's guide to the Soundex," *Journal of Family History*, **5**(1), pp. 112–115.

[130] Levenshtein, V. I. et al. (1966) "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, Soviet Union, pp. 707–710.

[131] Kochkina, E., M. Liakata, and A. Zubiaga "All-in-one: Multi-task Learning for Rumour Verification," in *ACL'18*.

[132] Gomez, R., J. Gibert, L. Gomez, and D. Karatzas (2020) "Exploring hate speech detection in multimodal publications," in *WACV'20*, pp. 1470–1478.

[133] Wulczyn, E., N. Thain, and L. Dixon (2017), "Wikipedia Talk Labels: Personal Attacks," .
URL `https://figshare.com/articles/dataset/Wikipedia_Talk_Labels_Personal_Attacks/4054689/6`

[134] Davidson, T., D. Warmsley, M. Macy, and I. Weber "Automated Hate Speech Detection and the Problem of Offensive Language," in *ICWSM'17*.

[135] Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019) "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*.

[136] Zeng, G., F. Qi, Q. Zhou, T. Zhang, B. Hou, Y. Zang, Z. Liu, and M. Sun (2021) "Openattack: An open-source textual adversarial attack toolkit," in *ACL'21, Demo*, pp. 363–371.

[137] Kim, Y. (2014) "Convolutional Neural Networks for Sentence Classification," in *EMNLP'14*, Doha, Qatar, pp. 1746–1751.

[138] Socher, R., A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts (2013) "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank," in *EMNLP'13*.

[139] Le, T., N. Park, and D. Lee "SHIELD: Defending Textual Neural Networks against Black-Box Adversarial Attacks with Stochastic Multi-Expert Patcher," *ACL'22*.

[140] Le, T., L.-T. Tran, and D. Lee "Socialbots on Fire: Modeling Adversarial Behaviors of Socialbots via Multi-Agent Hierarchical Reinforcement Learning," *WWW'22*.

Vita

Thai Le

Education

Aug 2017 – May 2022 The Pennsylvania State University (USA, PA) - *P.h.D* in Information Sciences & Technology
Aug 2021 – Apr 2015 Ritsumeikan Asia Pacific University (Japan, Oita) - *B.S*, International Management

Publication

[1] **[ACL'22a] Thai Le**, Noseong Park, Dongwon Lee. (2022). SHIELD: Defending Textual Neural Networks against Black-Box Adversarial Attacks with Stochastic Multi-Expert Patcher. 60th Annual Meeting of the Association for Computational Linguistics (ACL).

[2] **[ACL'22b] Thai Le**, Jooyoung Lee, Kevin Yen, Yifan Hu, Dongwon Lee. (2022). Perturbations in the Wild: Leveraging Human-Written Text Perturbations for Realistic Adversarial Attack and Defense. 60th Annual Meeting of the Association for Computational Linguistics (ACL) (Findings).

[3] **[WWW'22] Thai Le**, Long-Thanh Tran, Dongwon Lee (2022). Socialbots on Fire: Modeling Adversarial Behaviors of Socialbots via Multi-Agent Hierarchical Reinforcement Learning. The Web Conference (WWW).

[4] **[ACL'21] Thai Le**, Noseong Park, Dongwon Lee. (2021). A Sweet Rabbit Hole by DARCY: Using Honeypots to Detect Universal Trigger's Adversarial Attacks. 59th Annual Meeting of the Association for Computational Linguistics (ACL).

[5] **[ICDM'20] Thai Le**, Suhang Wang, Dongwon Lee. (2020). MALCOM: Generating Malicious Comments to Attack Neural Fake News Detection Models. 20th IEEE International Conference on Data Mining (ICDM).

[6] **[KDD'20] Thai Le**, Suhang Wang, Dongwon Lee. (2020). GRACE: Generating Concise and Informative Contrastive Sample to Explain Neural Network Model's Prediction. 26th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD).

[7] **[ASONAM'19] Thai Le**, Shu Kai, Maria Molina, Dongwon Lee, Shaym Sundar, Huan Liu. (2019). 5 Sources of Clickbaits You Should Know! Using Synthetic Clickbaits to Improve Prediction and Distinguish between Bot-Generated and Human-Written Headlines. In IEEE/ACM Int'l Conf. on Social Networks Analysis and Mining (ASONAM).