Towards Building a Robust Toxicity Predictor

Dmitriy Bespalov, Sourav Bhabesh, Yi Xiang, Liutong Zhou, Yanjun Qi
Amazon Web Services AI ML
{ dbespal, sbhabesh, yxxan, yanjunqi }@amazon.com

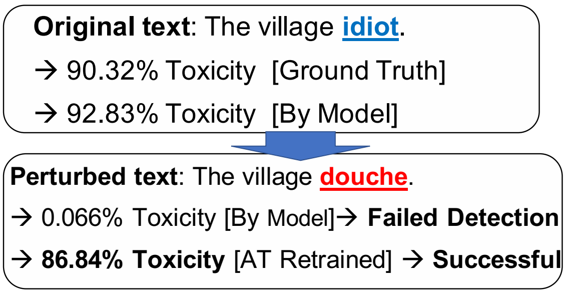
Abstract

Recent NLP literature pays little attention to the robustness of toxicity language predictors, while these systems are most likely to be used in adversarial contexts. This paper presents a novel adversarial attack, ToxicTrap, introducing small word-level perturbations to fool SOTA text classifiers to predict toxic text samples as benign. ToxicTrap exploits greedy based search strategies to enable fast and effective generation of toxic adversarial examples. Two novel goal function designs allow ToxicTrap to identify weaknesses in both multiclass and multilabel toxic language detectors. Our empirical results show that SOTA toxicity text classifiers are indeed vulnerable to the proposed attacks, attaining over 98% attack success rates in multilabel cases. We also show how a vanilla adversarial training and its improved version can help increase robustness of a toxicity detector even against unseen attacks.

1 Introduction

Deep learning-based natural language processing (NLP) plays a crucial role in detecting toxic language content (Ibrahim et al., 2018; Zhao et al., 2019; Djuric et al., 2015; Nobata et al., 2016; MacAvaney et al., 2019). Toxic content often includes abusive language, hate speech, profanity or sexual content. Recent methods have mostly leveraged transformer-based pre-trained language models (Devlin et al., 2019; Liu et al., 2019a) and achieved high performance in detecting toxicity (Zampieri et al., 2020). However, directly deploying NLP models could be problematic for real-world toxicity detection. This is because toxicity filtering is mostly needed in security-relevant industries like gaming or social networks where models are constantly being challenged by social engineering and adversarial attacks.

In this paper, we study the adversarial robustness of toxicity language predictors[1] and propose a new set of attacks, we call "ToxicTrap ". ToxicTrap generates targeted adversarial examples that fool a target model towards the benign predictions. Our design is motivated by the fact that most toxicity classifiers are being deployed as API services and used for flagging out toxic samples. Figure 1 shows one ToxicTrap adversarial example. The perturbed text replaces one word with its synonym and the resulting phrase fooled the transformer based detector into a failed detection (as "benign").



Original text: The village idiot.
→ 90.32% Toxicity [Ground Truth]
→ 92.83% Toxicity [By Model]

Perturbed text: The village douche.
→ 0.066% Toxicity [By Model]→ **Failed Detection**
→ **86.84% Toxicity** [AT Retrained] → **Successful**

Figure 1: ToxicTrap successfully fooled a SOTA toxicity predictor by perturbing one word in the original text using word synonym perturbation. After adversarial training (AT), the improved toxicity predictor can correctly flag the perturbed text into the toxicity class.

We propose novel goal functions to guide greedy word importance ranking to iteratively replace each word with small perturbations. Samples generated by ToxicTrap are toxic, and can fool a victim toxicity predictor model to classify them as "benign" and not as any toxicity classes or labels. The proposed ToxicTrap attacks can pinpoint the robustness of both multiclass and multilabel toxicity NLP models. To the authors' best knowledge, this paper is the first work that introduces adversarial attacks [2] to fool multilabel NLP tasks. Design multilabel ToxicTrap is challenging since coordinating multiple labels all at once and quantifying the attacking goals is tricky when aiming to multiple targeted

[1] We use “toxicity detection”, “toxicity language detection” and “toxicity prediction” interchangeably.

[2] This paper uses “methods for adversarial example generation” and “adversarial attacks” interchangeably.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 5: Industry Track, pages 581–598
July 10-12, 2023 ©2023 Association for Computational Linguistics

labels.

Empirically, we use ToxicTrap to evaluate BERT (Devlin et al., 2018) and DistillBERT (Liu et al., 2019b) based modern toxicity text classifiers on the Jigasw (Jig, 2018), and Offensive tweet (Bowman et al., 2015) datasets. We then use adversarial training to make these models more resistant to ToxicTrap adversarial attacks. In *adversarial training* a target model is trained on both original examples and adversarial examples (Goodfellow et al., 2014). We improve the vanilla adversarial training with an ensemble strategy to train with toxic adversarial examples generated from multiple attacks. Our contributions are as follows:

- ToxicTrap reveals that SOTA toxicity classifiers are not robust to small adversarial perturbations.
- Conduct a thorough set of analysis comparing variations of ToxicTrap designs.
- Empirically show that greedy unk search with composite transformation is preferred.
- Adversarial training can improve robustness of toxicity detector.

2 Method

Methods generating text adversarial examples introduce small perturbations in the input data checking if a target model's output changes significantly. These adversarial attacks help to identify if an NLP model is susceptible to word replacements, misspellings, or other variations that are commonly found in real-world data. For a given NLP classifier $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$ and a seed input $\mathbf{x}$, searching for an *adversarial* example $\mathbf{x}'$ from $\mathbf{x}$ is:

$$\begin{aligned} \mathbf{x}' = \mathcal{T}(\mathbf{x}, \Delta\mathbf{x}),\ \mathbf{x}' \in \mathcal{X} \\ \textbf{s.t.}\ \ \mathcal{G}(\mathcal{F}, \mathbf{x}'),\ \ \textbf{and}\ \ \{C_i(\mathbf{x}, \mathbf{x}')\} \end{aligned} \tag{1}$$

Here $\mathcal{T}(\mathbf{x}, \Delta\mathbf{x})$ denotes the transformations that perturb text $\mathbf{x}$ to $\mathbf{x}'$. $\mathcal{G}(\mathcal{F}, \mathbf{x}')$ represents a goal function that defines the purpose of an attack, for instance like flipping the output. $\{C_i(\mathbf{x}, \mathbf{x}')\}$ denotes a set of constraints that filters out undesirable $\mathbf{x}'$ to ensure that perturbed $\mathbf{x}'$ preserves the semantics and fluency of the original $\mathbf{x}$.

To solve Equation (1), adversarial attack methods design search strategies [3] to transform $\mathbf{x}$ to $\mathbf{x}'$ via transformation $\mathcal{T}(\mathbf{x}, \Delta\mathbf{x})$, so that $\mathbf{x}'$ fools $\mathcal{F}$ by achieving the fooling goal $\mathcal{G}(\mathcal{F}, \mathbf{x}')$, and at the same time fulfilling a set of constraints $\{C_i(\mathbf{x}, \mathbf{x}')\}$. Therefore designing adversarial attacks focus on designing four components: (1) goal function, (2) transformation, (3) search strategy, and (4) constraints between seed and its adversarial examples (Morris et al. (2020b)).

We propose a suite of ToxicTrap attacks to identify vulnerabilities in SOTA toxicity NLP detectors. ToxicTrap attacks focus on intentionally generating perturbed texts that contain the same highly abusive language as original toxic text, yet receive significantly lower toxicity scores and get predicted as "benign" by a target model.

2.1 Word transformations in ToxicTrap

For $\mathcal{T}(\mathbf{x}, \Delta\mathbf{x})$, many possible word perturbations exist, including embedding based word swap, thesaurus based synonym substitutions, or character substitution (Morris et al., 2020a).

Attacks by Synonym Substitution: Our design focuses on transformations that replace words from an input with its synonyms.

- (1) Swap words with their N nearest neighbors in the counter-fitted GloVe word embedding space; where $N \in \{5, 20, 50\}$.
- (2) Swap words with those predicted by BERT Masked Language Model (MLM) model.
- (3) Swap words with their nearest neighbors in the wordNet.

The goal of word synonym replacement is to create examples that can preserve semantics, grammaticality, and non-suspicion.

Attacks by Character Transformation: Another group of word transformations is to generate perturbed word via character manipulations. This includes character insertion, deletion, neighboring character swap and/or character substitution by Homoglyph. These transformations change a word into one that a target toxicity detection model doesn't recognize. These character changes are designed to generate character sequences that a human reader could easily correct into those original words. Language semantics are preserved since human readers can easily correct the misspellings.

Composite Transformation: We also propose to combine the above transformations to create new composite transformations. For instance, one composite transformation can include both perturbed words from character substitution by Homoglyph and from word swaps using nearest neighbors from GloVe embedding (Pennington et al., 2014).

[3] Because brute force is not feasible considering the length and dictionary size of natural language text.

2.2 Novel goal functions for ToxicTrap

A goal function $\mathcal{G}(\mathcal{F}, \mathbf{x}')$ module defines the purpose of an attack. Different kinds of goal functions exist in the literature to define whether an attack is successful in terms of a victim model's outputs.

Toxicity language detection has centered on a supervised classification formulation, including binary toxicity detector, multiclass toxicity classification and multilabel toxicity detection which assigns a set of target labels for $\mathbf{x}$. See Section A.2 for more details. As aforementioned, toxicity classifiers are used mostly as API services to filter out toxic text. Therefore, its main vulnerability are those samples that should get detected as toxic, however, fooling detectors into a wrong prediction as "benign".

Now let us define $\mathcal{G}(\mathcal{F}, \mathbf{x}')$ for ToxicTrap attacks. We propose two choices of designs regarding the target model types.

Multiclass or Binary Toxicity: When toxicity detector $\mathcal{F}$ handles binary or multiclass outputs, we define ToxicTrap attacks' goal function as:

$$\mathcal{G}(\mathcal{F}, \mathbf{x}') := \{\mathcal{F}(\mathbf{x}') = b; \mathcal{F}(\mathbf{x}) \neq b\} \tag{2}$$

Here b denotes the "0:benign" class.

Multilabel Toxicity: Next, we study how to fool multilabel toxicity predictors. In real-world applications of toxicity identification, an input text may associate with multiple labels, like identity attack, profanity, and hate speech at the same time. The existence of multiple toxic labels at the same time provides more opportunities for attackers, but also poses design challenges [4].

Multilabel toxicity detection assigns a set of target labels for $\mathbf{x}$. The output $\mathbf{y} = \{y_1, y_2, ..., y_L\}$ is a vector of L binary labels and each $y_i \in \{0, 1\}$ (Zhao et al., 2019). For example in the Jigsaw dataset (Jig, 2018), each text sample associates with six binary labels per sample, namely {benign, obscene, identity attack, insult, threat, and sexual explicit}. We introduce a novel goal function for attacking such models as follows:

$$\begin{aligned}\mathcal{G}(\mathcal{F}, \mathbf{x}') := \{&\mathcal{F}_b(\mathbf{x}') = 1; \mathcal{F}_b(\mathbf{x}) = 0; \\ &\mathcal{F}_t(\mathbf{x}') = 0, \forall t \in \mathbf{T}\}\end{aligned} \tag{3}$$

Here, $\mathbf{T} = \{y_2, ..., y_L\}$ denotes the set of toxic labels, and $b = \{y_1 : \text{Benign}\}$ is the non-toxic or benign label. $\{\mathcal{F}_b(\mathbf{x}') = 1; \mathcal{F}_b(\mathbf{x}) = 0\}$ denotes "$\mathbf{x}'$ gets predicted as Benign, though $\mathbf{x}$ is toxic". And $\{\mathcal{F}_t(\mathbf{x}') = 0, \forall t \in \mathbf{T}\}$ denotes $\mathbf{x}'$ is not predicted as any toxicity types. In summary, our ToxicTrap attacks focus on perturbing correctly predicted toxic samples.

2.3 Language constraints in ToxicTrap

In Equation (1), we use a set of language constraints to filter out undesirable $\mathbf{x}'$ to ensure that perturbed $\mathbf{x}'$ preserves the semantics and fluency of the original $\mathbf{x}$, and with as fewer perturbations as possible. There exist a variety of possible constraints in the NLP literature (Morris et al., 2020b). In ToxicTrap , we decide to use the following list:

- Limit on the ratio of words to perturb to 10%
- Minimum angular similarity from universal sentence encoder (USE) is 0.84
- Part-of-speech match.

2.4 Greedy Search strategies in ToxicTrap

Solving Equation (1) is a combinatorial search task searching within all potential transformations to find those transformations that result with a successful adversarial example, aka, achieving the fooling goal function and satisfying the constraints. Due to the exponential nature of search space, many heuristic search algorithms existed in the recent literature, including like greedy search, beam search, and population-based search (Zhang et al., 2019).

For a seed text $\mathbf{x} = (w_1, \ldots, w_i, \ldots, w_n)$, a perturbed text $\mathbf{x}'$ can be generated by swapping w_i with altered w_i'. The main role of a search strategy is to decide what word w_i from $\mathbf{x}$ to perturb next. We propose to center the search design of ToxicTrap using greedy search with word importance ranking to iteratively replace one word at a time to generate adversarial examples.

The main idea is that words of $\mathbf{x}$ are first ranked according to an importance function. Four possible choices: (1) "unk " based: word's importance is determined by how much a heuristic score (details later) changes when the word is substituted with an UNK token. (2) "delete ": word's importance is determined by how much the heuristic score changes when the word is deleted from the original input. (3) "weighted saliency" or wt-saliency : words are ordered using a combination of the change in score when the word is substituted with an UNK token multiplied by the maximum score gained by perturbing the word. (4) "gradient ": each word's importance is calculated using the gradient of the

[4] To the authors' best knowledge, no multilabel adversarial attacks exist in the NLP literature.

victim's loss with respect to the word and taking its L1 norm as the word' importance.

After words in $\mathbf{x}$ get sorted in the order of descending importance, word w_i is then substituted with w'_i with allowed transformation. This is until the fooling goal is achieved, or the number of perturbed words reaches upper bound. The heuristic scoring function used in the word importance ranking relates to the victim model and the fooling goal. For instance, when we work with a binary toxicity model, the heuristic score `ToxicTrap` equals to the model's output score for the target "0:benign" class. Algorithm 1 shows our design of the score function for the multilabel `ToxicTrap` attacks. In experiments, we also evaluate two other search strategies: "`beam` search" and "`genetic` search". Our empirical results indicate strong performance from the greedy search with word importance ranking over other strategies.

Algorithm 1 provides the pseudo code of how we implement Equation (3) as a new goal function in the TextAttack python library (Morris et al., 2020a). The implemented *MultilabelClassificationGoalFunction* extends the TextAttack library to multilabel tasks.

2.5 Harden with Adversarial Training

Our ultimate goal of designing `ToxicTrap` attacks is to improve toxicity NLP models' adversarial robustness. A simple strategy called Adversarial Training (`AT`) has been a major defense strategy for improving adversarial robustness (Madry et al., 2018). The vanilla adversarial training process involves augmenting the training data with adversarial examples generated from perturbing the training data in the input space. Two variations of adversarial training exist. (1) If the augmented adversarial examples are generated from a single attack approach, we name this process as `AT1` . (2) If the augmented examples are generated from multiple attack methods, we call the training as `AT2` .

2.6 `ToxicTrap` Recipes and Extensions

The modular design of `ToxicTrap` allows us to implement many different `ToxicTrap` attack recipes in a shared framework, combining different goal functions, constraints, transformations and search strategies. In Section 3, we conduct a thorough empirical analysis to compare possible `ToxicTrap` recipes and recommend `unk` greedy search and the composite transformation for most use cases. Table 1 lists our recommended recipe.

Algorithm 1 Attack with *MultilabelClassificationGoalFunction*

```
Input: An original text $\mathbf{x}$, a multilabel classifier $\mathcal{F}$,
    a set of targeted labels as $T$ (for which scores are to be maximized),
    a set of other labels as $N$ (for which scores are to be minimized);
    maximization threshold $\epsilon_{\text{maximize}} = 0.5$; search method
    $S() :=$ Greedy-WIR, transformations $\mathcal{T}$, a set of constraints
    as $\mathbf{C}$, and the max number of trials $I$.
Output: adversarial example $x'$, Attack Status = {Fail, Success}
Initialize $x' \leftarrow None$, goal $\leftarrow -\infty$, $\epsilon_{\text{minimize}} = 1 - \epsilon_{\text{maximize}}$
for trial $i = 1, \ldots, I$ do
  $\tilde{x} \leftarrow \mathcal{T}(x,\ S(\text{goal}, x, i))$
  if $\forall C \in \mathbf{C}$, $C(x, \tilde{x})$ is not True then
    Continue
  end if
  scores $\leftarrow sigmoid(\ \mathcal{F}(\tilde{x})\ )$
  goal$'$ $\leftarrow \sum_{l \in L_{\text{max}}}$ scores$[l]$ $+ \sum_{l \in L_{\text{min}}}(1 -$ scores$[l])$
  if goal$'$ $>$ goal then
    goal $\leftarrow$ goal$'$   # search $S()$ will use goal value
    if scores$[l] > \epsilon_{\text{maximize}}$ for $\forall l \in T$ and
       scores$[l] < \epsilon_{\text{minimize}}$ for $\forall l \in N$ then
      $x' \leftarrow \tilde{x}$
      return $x'$, Attack Succeeded
    end if
  end if
end for
return $x'$, Attack Failed
```

Besides, we select to adapt another five SOTA combinations of transformation and constraints from popular general NLP adversarial example recipes in the literature to create `ToxicTraps Extend` attacks in Table 14, covering a good range of transformations and constraints. Table 15 shows generated adversarial examples using these attacks.

3 Experiments

We conducted a series of experiments covering three different toxicity classification tasks: binary, multilabel, and multiclass; over two different transformer architectures: DistillBERT and BERT; and across two datasets: the large-scale Wikipedia Talk Page dataset- Jigsaw (Jig, 2018) and the Offensive Tweet for hate speech detection dataset (Davidson

Attack Recipe	Constraints	Transformation	Search Method
ToxicTrap general recommendation	USE sentence encoding angular similarity > 0.84, Part-of-speech match, Ratio of number of words modified < 0.1	{Random Character Insertion, Random Character Deletion, Neighboring Character Swap, Character Substitution by Homoglyph, Word Synonym Replacement with 20 nearest neighbors in the counter-fitted GLOVE word embedding space}	Greedy word importance ranking (unk -based)

Table 1: Recommended ToxicTrap Attack Recipe

Search	unk	unk	unk	unk	unk
POS		x	x	x	x
Transformation (N)	glove (20) + chars	glove (20) + chars	wordnet (-)	mlm (20)	glove (50)
Seed Text					
thank god FOB didnt come back doing some **shitty** arena tour...	(0.911) thank god FOB didnt come back doing some **bollocks** arena tour...	(0.906) thank god FOB didnt come back doing some **lousy** arena tour...	(0.917) thank god FOB didnt come back doing some **stinky** arena tour...	(0.908) thank god FOB didnt come back doing some **wild** arena tour...	(0.910) thank god FOB didnt come back doing some **foul** arena tour...
Jews and white race **traitors** have to be...	(0.588) Jews and white race **traitors** have to be ...	(0.588) Jews and white race **traitors** have to be...	(0.507) Jews and white race **betrayer** have to be...	(0.557) Jews and white race **members** have to be...	(0.477) Jews and white race **betrayers** have to be...
Fluffy is a **retard** homeboy is calling a baby...	(0.566) Fluffy is a **retarded** homeboy is calling a baby...	(0.520) Fluffy is a **moron** homeboy is calling a baby...	(0.520) Fluffy is a **moron** homeboy is calling a baby...	(0.915) Fluffy is a **child** homeboy is calling a baby...	(0.520) Fluffy is a **moron** homeboy is calling a baby...
ugh your with Tebow? You're officially a **fag** now and in followed.	(0.897) ugh your with Tebow? You're officially a **poof** now and in followed.	(0.897) ugh your with Tebow? You're officially a **poof** now and in followed.	(0.908) ugh your with Tebow? You're officially a **fatigue** now and in followed.	(0.907) ugh your with Tebow? You're officially a **wolf** now and in followed.	(0.907) ugh your with Tebow? You're officially a **poofter** now and in followed.

Table 2: Selected toxic adversarial examples generated attacking HTweet-MC model. Perturbed scores are reported in parenthesis. Adversarial examples were generated using unk search method; with and without POS constraint; and using three word synonym substitution transformations with number of nearest neighbors specified in parenthesis; chars indicates that character transformations were applied. More in Table 9.

et al., 2017). Table 3 lists two datasets' statistics. Section A.4 provides more details.

Base Toxicity Models: Our experiments work on three base models, including {Jigsaw-BL , Jigsaw-ML , HTweet-MC } to cover three types of toxicity prediction tasks. See details in Section A.5.

Implementation: We implement all of our ToxicTrap and ToxicTraps Extend attacks using the NLP attack package TextAttack[5] (Morris et al., 2020a). When generating adversarial examples, we only attack seed samples that are correctly predicted by a victim model. (Adversarial attack does not make sense if the target model could not predict the seed sample correctly!). In our setup, this means we only use toxic seed samples when attacking three base models. This set up simulates real-world situations in which people intentionally create creative toxic examples to circumvent toxicity detection systems.

Evaluation Metrics: We use attack success rate ($ASR = \frac{\text{\# of successful attacks}}{\text{\# of total attacks}}$) to measure how successful each ToxicTrap attacking a victim model. To measure the runtime of each algorithm, we use the average number of queries to the victim model as a proxy. We also report the average percentage of words perturbed from an attack. In addition, for models trained with adversarial training, we report both, the model prediction performance and model robustness (by attacking robust model again).

3.1 Results on Attacking 3 Toxicity Predictors

Table 2 provides a few selected adversarial examples generated by attacking HTweet-MC model with five variations of ToxicTrap . The first column provides seed text that was used to generate adversarial examples.

Several observations can be made when comparing these examples. It is important to use POS constraint to generate syntactically correct examples. For the first seed text, a recipe without POS constraint (second column) produced replacement word "bullocks", while including POS constraint in the recipe (third column) produced syntactically correct example with replacement word "lousy". We also see that using glove for word synonym

[5]TextAttack https://github.com/QData/TextAttack.

Dataset	Train	Dev	Test	Test Toxic Samples	Train Toxic Samples
Jigsaw (Jig, 2018)	1.48MM	185k	185k	8,909	71,273
Offensive Tweet (Davidson et al., 2017)	20k	2.2k	2.5k	1897 (offensive) + 145 (hateful)	15510 + 1142

Table 3: Overview of the data statistics.

Base Model ⟶	`Jigsaw-BL`	`Jigsaw-ML`	`HTweet-MC`
Dataset	Jigsaw	Jigsaw	HateTweet
Classification	Binary	Multilabel	Multiclass
Architecture	DistillBERT	DistillBERT	BERT
LearningRate	2.06e-05	3.80e-05	2.66e-05
Epochs	5	10	10

Table 4: Overview of the base model statistics.

Task	Search	POS	Attack Success Rate	Average Number of Queries	Average Perturbed Word %
`Jigsaw-BL`	`gradient`		98.74	34.68	7.58
	`gradient`	x	98.72	26.78	7.06
	`delete`		99.42	55.38	7.38
	`delete`	x	99.21	48.03	6.80
	`unk`		99.32	55.11	7.12
	`unk`	x	99.27	47.62	6.76
	`wt-saliency`	x	99.19	407.43	6.71
	`genetic`	x	92.67	846.41	8.73
	`beam`	x	99.68	658.55	6.95
`Jigsaw-ML`	`gradient`		97.62	38.45	8.36
	`gradient`	x	97.72	29.78	7.56
	`delete`		98.71	57.60	7.54
	`delete`	x	98.63	49.93	6.99
	`unk`		98.75	57.08	7.70
	`unk`	x	98.75	49.38	6.96
	`wt-saliency`	x	98.51	419.58	6.91
	`genetic`	x	88.91	876.81	8.82
	`beam`	x	99.54	756.05	7.19
`HTweet-MC`	`gradient`		67.16	63.51	24.13
	`gradient`	x	67.56	49.67	24.08
	`delete`		71.46	58.78	18.96
	`delete`	x	71.46	48.17	19.37
	`unk`		72.38	58.99	18.80
	`unk`	x	72.23	48.04	19.43
	`wt-saliency`	x	74.71	178.91	18.81
	`genetic`	x	80.49	1025.66	21.86
	`beam`	x	90.07	442.18	18.76

Table 5: Effect of different search strategies on attack performance. Search column identifies type of search method. POS column identifies if part-of-speech matching constraint is used. The composite transformation is used: `glove` with $N = 20$ plus the character transformations. Results from rows on "`unk` + POS" can compare with "`unk` " rows in Table 6.

substitution is a better choice than `mlm` or `wordnet` . For the second and third seed text, `mlm` did not generate toxic phrases. In addition, we see that the recipe using `glove` (50) (last column) often generates similar examples as the `glove` (20) (third column). Finally, we observe that using character manipulation can generate adversarial examples with the same toxic meaning that fool the classifier.

Task	Transformation	N	Attack Success Rate	Average Number of Queries	Average Perturbed Word %
`Jigsaw-BL`	`wordnet`	-	89.59	34.87	6.66
	`glove`	5	86.04	30.55	7.06
	`glove`	20	96.75	41.87	6.68
	`glove`	50	98.38	64.17	6.55
	`mlm`	20	93.29	39.57	6.55
`Jigsaw-ML`	`wordnet`	-	87.72	37.01	6.81
	`glove`	5	84.33	32.15	7.47
	`glove`	20	95.92	43.84	6.89
	`glove`	50	97.91	67.09	6.68
	`mlm`	20	92.73	41.33	6.62
`HTweet-MC`	`wordnet`	-	56.66	32.08	17.94
	`glove`	5	33.65	21.82	23.06
	`glove`	20	66.70	41.06	18.85
	`glove`	50	69.99	81.14	18.18
	`mlm`	20	65.23	33.66	21.30

Table 6: Comparing synonym transformations only. No character transformations used. Reporting attack performance when using `unk` greedy search. The same constraints as in Table 5 with POS (part-of-speech) match.

Training	AUC	AP	F1	Recall
No AT	0.935	0.786	0.73	0.71
`AT1-delete`	0.936	0.792	0.74	0.719
`AT1-unk`	0.938	0.785	0.738	0.723
`AT2`	0.932	0.778	0.685	0.641

Table 7: Effect of adversarial training on model performance. Macro-average metrics for `HTweet-MC` model.

For the second seed text, character transformation (second and third columns) generates replacement word "traitors" where the second "t" is replaced with a monospace Unicode character "`t`".

3.2 Comparing Constraints in `ToxicTrap`

Then we study the effect of the part-of-speech match (POS) constraint on the attack performance. Table 5 shows that the use of POS constraint lowers average number of queries sent to the victim model. We observe this phenomena across all three tasks and all three search methods (`gradient` , `delete` , `unk`). For example, when attacking `Jigsaw-ML` model using `unk` with and without POS constraint, average number of queries are 49.38 and 57.08, respectively. We also observe that for most of the recipes, using POS constraint slightly decreases attack success rate (ASR). Considering the empirical results in Table 5 and the anecdotal examples in Table 2, we conclude POS constraint is preferred.

Training	Search	Attack Success Rate	Average Number of Queries	Average Perturbed Word %
No AT	delete	71.46	48.17	19.37
No AT	unk	72.23	48.04	19.43
AT1-delete	delete	21.97	69.68	28.83
AT1-unk	unk	11.17	74.71	33.07
AT2	delete	8.28	75.71	27.89
AT2	unk	6.08	77.91	33.24

Table 8: Effect of adversarial training on attack performance. Results for HTweet-MC model are reported. When attacking, ToxicTrap uses glove with $N = 20$ plus character transformations; constraint with POS; and search with two different greedy search methods.

3.3 Comparing Search in ToxicTrap

Table 5 also compares the effect of using different search methods on the attack performance. It shows that a greedy search method is preferred over genetic and beam . For example, when compared to unk , genetic and beam require almost 10x as many queries on average for all three tasks. The beam search results in higher ASR values on all three tasks, while genetic only outperforms greedy methods when attacking HTweet-MC . In addition, attacking Jigsaw-BL and Jigsaw-ML , beam only slightly outperforms greedy methods. Among the greedy search methods, unk is a good choice, as it provides consistently good ASR performance on all three tasks. It is worth noting that unk outperforms other three greedy search methods, except for wt-saliency when attacking HTweet-MC . However, attacking HTweet-MC model with wt-saliency requires more than 3x as many queries as the unk method.

3.4 Comparing Synonym Transformations

Now we compare word synonym substitutions, when the unk search method is used and the character manipulation are not (to single out the effect). Table 6 shows that glove with $N = 20$ nearest neighbors is an optimal choice for all three tasks. We observe that the wordnet and mlm transformations result in lower ASRs than glove . Also, glove with $N = 50$ only slightly lifts ASRs when compared to glove with $N = 20$. At the same time, using $N = 50$ nearest neighbors sends over 50% more queries to the victim models. We include the analysis of using different transformations with three different search methods (delete , unk , wt-saliency) in the Appendix in Table 10. These results also confirm that the choice of glove with $N = 20$ is preferred.

3.5 Results from Adversarial Training

Empirically, we explore how AT1 and AT2 impact both prediction performance and adversarial robustness. Table 7 and Table 8 present AT results on the HTweet-MC task (Table 12 and Table 13 on two other tasks). In Table 7, we observe that AT1-delete and AT1-unk both maintain the regular prediction performance as the base model. Table 8 shows the attack success metrics when we use ToxicTrap to attack the retrained robust HTweet-MC models. The AT1 models trained from using AT1-delete and AT1-unk attacks show significant improvements in robustness after AT1 adversarial training. We recommend readers to use AT1-unk as their default choice for hardening general toxic language predictors, since in both tables, AT1-unk outperforms AT1-delete slightly.

For the AT2 robust model, ToxicTrap attacks are "unseen" (we used the five ToxicTraps Extend attacks from Section B to create the AT2 model in our experiments). Our results show AT2 can harden HTweet-MC model not only against attacks it is trained on (ToxicTraps Extend) but also against attacks it has not seen before (ToxicTrap). This could be attributed to the hypothesis that an unseen attack may share similar underlying patterns with the attack ensemble that AT2 model has used. In Table 7, AT2 slightly under-performs base on regular predictions, since it was trained with more adversarial examples from multiple attacks.

4 Conclusion

Text toxicity prediction models are not designed to operate in the presence of adversaries. This paper proposes a suite of ToxicTrap attacks to identify weaknesses in SOTA toxicity language predictors that could potentially be exploited by attackers. We also evaluate how adversarial training improves model robustness across seen and unseen attacks. As next steps, we plan to investigate other strategies like virtual adversarial training, disentangled representation learning or generative methods and pinpoint how they will influence the robustness of toxicity predictors (Qiu et al., 2022).

References

2018. Toxic comment classification challenge. https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge.

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

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On evaluating adversarial robustness. *CoRR*, abs/1902.06705.

Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. 2018. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *CoRR*, abs/1803.01128.

Home Affairs Committee et al. 2017. Hate crime: Abuse, hate and extremism online. *Fourteenth Report of Session 2016*, 17.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

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

Nemanja Djuric, Jing Zhou, Robin Morris, Mihajlo Grbovic, Vladan Radosavljevic, and Narayan Bhamidipati. 2015. Hate speech detection with comment embeddings. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 29–30, New York, NY, USA. Association for Computing Machinery.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. In *ACL*.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56.

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

Sreyan Ghosh and Sonal Kumar. 2021. Cisco at semeval-2021 task 5: What's toxic?: Leveraging transformers for multiple toxic span extraction from online comments.

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

Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. 2022. Toxigen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection. *arXiv preprint arXiv:2203.09509*.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective API built for detecting toxic comments. *CoRR*, abs/1702.08138.

Mai Ibrahim, Marwan Torki, and Nagwa El-Makky. 2018. Imbalanced toxic comments classification using data augmentation and deep learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 875–878.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is bert really robust? natural language attack on text classification and entailment. *ArXiv*, abs/1907.11932.

NF Johnson, R Leahy, N Johnson Restrepo, N Velasquez, M Zheng, P Manrique, P Devkota, and Stefan Wuchty. 2019. Hidden resilience and adaptive dynamics of the global online hate ecology. *Nature*, 573(7773):261–265.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. Textbugger: Generating adversarial text against real-world applications. *ArXiv*, abs/1812.05271.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Sean MacAvaney, Hao-Ren Yao, Eugene Yang, Katina Russell, Nazli Goharian, and Ophir Frieder. 2019. Hate speech detection: Challenges and solutions. *PLoS ONE*, 14:1–16.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*.

John Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. 2020a. TextAttack: A framework for adversarial attacks in natural language processing. *ArXiv*, abs/2005.05909.

John X. Morris, Eli Lifland, Jack Lanchantin, Yangfeng Ji, and Yanjun Qi. 2020b. Reevaluating adversarial examples in natural language.

Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. Abusive language detection in online user content. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 145–153, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Shilin Qiu, Qihe Liu, Shijie Zhou, and Wen Huang. 2022. Adversarial attack and defense technologies in natural language processing: A survey. *Neurocomputing*, 492:278–307.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Zhongguo Wang and Bao Zhang. 2021. Toxic comment classification based on bidirectional gated recurrent unit and convolutional neural network. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 21(3).

Tong Xiang, Sean MacAvaney, Eugene Yang, and Nazli Goharian. 2021. Toxccin: Toxic content classification with interpretability.

Jin Yong Yoo and Yanjun Qi. 2021. Towards improving adversarial training of nlp models.

Marcos Zampieri, Preslav Nakov, Sara Rosenthal, Pepa Atanasova, Georgi Karadzhov, Hamdy Mubarak, Leon Derczynski, Zeses Pitenis, and Çağrı Çöltekin. 2020. SemEval-2020 task 12: Multilingual offensive language identification in social media (OffensEval 2020). In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 1425–1447, Barcelona (online). International Committee for Computational Linguistics.

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

Wei Emma Zhang, Quan Z. Sheng, and Ahoud Abdulrahmn F. Alhazmi. 2019. Generating textual adversarial examples for deep learning models: A survey. *CoRR*, abs/1901.06796.

Zhixue Zhao, Ziqi Zhang, and Frank Hopfgartner. 2019. Detecting toxic content online and the effect of training data on classification performance. *EasyChair Preprints*.

A Appendix

A.1 Related Works

To the authors' best knowledge, the toxicity NLP literature includes very limited studies on adversarial attacks. Only one study from Hosseini et al. (2017) tried to deceive Google's perspective API for toxicity identification by misspelling the abusive words or by adding punctuation between the letters. This paper tries to conduct a comprehensive study by introducing a wide range of novel attack recipes and improving adversarial training to enable robust text toxicity predictors.

Next, we also want to point out existing studies on text adversarial examples center in generating adversarial examples against binary and multi-class classification models (Morris et al., 2020a). To the authors' best knowledge, no multilabel adversarial attacks exist in the NLP literature. Our work is the first that designs novel attacks against the multilabel toxicity predictors. The design of multilabel adversarial examples is challenging since coordinating multiple labels all at once and quantifying the attacking goals is tricky because it is harder to achieve multiple targeted labels. Simply adapting attacks for binary or multiclass models (Morris et al., 2020a) to multilabel setup is not feasible. In multilabel prediction, each instance can be assigned to multiple labels. This is different from the multi-class setting in which classes are mutually exclusive and one sample can only associate to one class (label). The existence of multiple labels at the same time provides better opportunities for attackers, but also posts design challenges. Our design in Equation (3) and Algorithm 1 has paved a path for multilabel text adversarial example research.

A.2 Toxicity Detection from Text

The mass growth of social media platforms has enabled efficient exchanges of opinions and ideas between people with diverse background. However, this also brings in risk of user generated toxic contents that may include abusive language, hate speech or cyberbullying. Toxic content may lead to incidents of hurting individuals or groups (Johnson et al., 2019), calling for automated tools to detect toxicity for maintaining healthy online communities.

Automatic content moderation uses machine learning techniques to detect and flag toxic content It is critical for online platforms to prohibit toxic language, since such content makes online communities unhealthy and may even lead to real crimes (Johnson et al., 2019; Committee et al., 2017).

Past literature on toxicity language detection has centered on a supervised classification formulation (Zhao et al., 2019; Djuric et al., 2015; Nobata et al., 2016; MacAvaney et al., 2019). We denote $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$ as a supervised classification model, for example, a deep neural network classifier. $\mathcal{X}$ denotes the input language space and $\mathcal{Y}$ represents the output space. For a sample $(\mathbf{x}, \mathbf{y})$, $\mathbf{x} \in \mathcal{X}$ denotes the textual content [6] and $\mathbf{y} \in \mathcal{Y}$ denotes its toxicity label(s). The toxicity detection task varies with what $\mathbf{y}$ stands for. The literature has included three main cases:

(1) binary toxicity detector, here $\mathbf{y}$ from $\{0$: benign,1:toxic$\}$;

(2) multilabel toxicity detection which assigns a set of target labels for $\mathbf{x}$. Here $\mathbf{y} = \{y_1, y_2, ..., y_L\}$ is a vector of L binary labels and each $y_i \in \{0, 1\}$ (Zhao et al., 2019). For example in the Jigsaw dataset (Jig, 2018), each text sample associates with six binary labels per sample, namely {benign, obscene, identity attack,insult,threat, and sexual explicit};

(3) multiclass toxicity classification, and $\mathbf{y}$ is a discrete integer. For example, the Offensive Tweet dataset (Davidson et al., 2017) has three classes - {0:benign, 1:offensive, and 2:hate} (one class per sample).

The literature on toxicity has been mostly focused on improving accuracy, via feature engineering (Zampieri et al., 2020), deep representation learning (Wang and Zhang, 2021) and via fine tuning from pretrained large language models (Ghosh and Kumar, 2021). Recently literature has extended to investigate these classifiers' interpretability (Xiang et al., 2021) and fairness (Hartvigsen et al., 2022).

A.3 Basics of Text Adversarial Examples

Research has shown that current deep neural network models lack the ability to make correct predictions on adversarial examples (Szegedy et al., 2014). The field of investigating the adversarial robustness of NLP models has seen growing interest,

[6] this paper focuses on toxicity detection from text

Search	unk	unk	unk	unk	unk
POS		x	x	x	x
Transformation (N)	glove (20) + chars	glove (20) + chars	wordnet (-)	mlm (20)	glove (50)
Seed Text					
My neighbors hate me cuz i smoke **weed** and blast the most ghetto music f...	(0.554) My neighbors hate me cuz i smoke **pot** and blast the most ghetto music from...	(0.554) My neighbors hate me cuz i smoke **pot** and blast the most ghetto music from...	(0.554) My neighbors hate me cuz i smoke **pot** and blast the most ghetto music from...	(0.554) My neighbors hate me cuz i smoke **pot** and blast the most ghetto music from...	(0.554) My neighbors hate me cuz i smoke **pot** and blast the most ghetto music from...
thank god FOB didnt come back doing some **shitty** arena tour...	(0.911) thank god FOB didnt come back doing some **bollocks** arena tour...	(0.906) thank god FOB didnt come back doing some **lousy** arena tour...	(0.917) thank god FOB didnt come back doing some **stinky** arena tour...	(0.908) thank god FOB didnt come back doing some **wild** arena tour...	(0.910) thank god FOB didnt come back doing some **foul** arena tour...
Jews and white race **traitors** have to be swept clean so the Aryan race ca...	(0.588) Jews and white race **traitors** have to be swept clean so the Aryan race can...	(0.588) Jews and white race **traitors** have to be swept clean so the Aryan race can...	(0.507) Jews and white race **betrayer** have to be swept clean so the Aryan race can...	(0.557) Jews and white race **members** have to be swept clean so the Aryan race can ...	(0.477) Jews and white race **betrayers** have to be swept clean so the Aryan race ca...
How ghetto **girls** pronounce there names... I could cry right now oh my g...	(0.474) How ghetto **females** pronounce there names... I could cry right now oh my g...	(0.474) How ghetto **females** pronounce there names... I could cry right now oh my g...	(0.588) How ghetto **daughter** pronounce there names... I could cry right now oh my ...	(0.534) How ghetto **ers** pronounce there names... I could cry right now oh my god h...	(0.474) How ghetto **females** pronounce there names... I could cry right now oh my g...
America is **white** trash and so are all Americans.	(0.863) America is **white** trash and so are all Americans.	(0.863) America is **white** trash and so are all Americans.	(0.899) America is **ashen** trash and so are all Americans.	(0.784) America is **utter** trash and so are all Americans.	(0.897) America is **pristine** trash and so are all Americans.
Fluffy is a **retard** homeboy is calling a baby rattlesnake friend and kis...	(0.566) Fluffy is a **retarded** homeboy is calling a baby rattlesnake friend and kis...	(0.520) Fluffy is a **moron** homeboy is calling a baby rattlesnake friend and kissin...	(0.520) Fluffy is a **moron** homeboy is calling a baby rattlesnake friend and kissin...	(0.915) Fluffy is a **child** homeboy is calling a baby rattlesnake friend and kissin...	(0.520) Fluffy is a **moron** homeboy is calling a baby rattlesnake friend and kissin...
More **queer** lady Templars. That's my point.	(0.908) More **fairy** lady Templars. That's my point.	(0.908) More **fairy** lady Templars. That's my point.	(0.910) More **curious** lady Templars. That's my point.	(0.901) More **likely** lady Templars. That's my point.	(0.910) More **cheerful** lady Templars. That's my point.
ugh your with Tebow? You're officially a **fag** now and in followed.	(0.897) ugh your with Tebow? You're officially a **poof** now and in followed.	(0.897) ugh your with Tebow? You're officially a **poof** now and in followed.	(0.908) ugh your with Tebow? You're officially a **fatigue** now and in followed.	(0.907) ugh your with Tebow? You're officially a **wolf** now and in followed.	(0.907) ugh your with Tebow? You're officially a **poofter** now and in followed.

Table 9: Selected Toxic Adversarial Examples. Here we only show adversarial examples generated by attacking base model HTweet-MC on Offensive Tweet text, since it contains much shorter messages than Jigsaw. Perturbed scores for adversarial examples are reported in parenthesis. Adversarial examples were generated using unk greedy search method. POS row marked with an "x" indicates that the part-of-speech matching constraint was used. Transformation row indicates which word substitution method was used (glove , wordnet , mlm), and number of nearest neighbors N is specified in parenthesis. chars indicates that character transformations were applied.
We used different word transformations for synonym substitution with varying number of nearest neighbors (20 or 50). Two recipes used character transformations, while the other three did not. Also, one recipe did not use part-of-speech match (POS) constraint, and it was included in the rest of recipes. All five recipes used unk greedy search method.

with a body of new adversarial attacks[7] designed to fool question answering (Jia and Liang, 2017), machine translation (Cheng et al., 2018), and text classification systems (Ebrahimi et al., 2017; Jia and Liang, 2017; Alzantot et al., 2018; Jin et al., 2019; Ren et al., 2019; Zang et al., 2020; Garg and Ramakrishnan, 2020).

A.4 Datasets Details

A.4.1 Jigsaw.

This dataset was derrived from the Wikipedia Talk Pages dataset published by Google and Jigsaw on Kaggle (Jig, 2018). Wikipedia Talk Page allows users to discuss improvements to articles via comments. The comments are anonymized and labeled with toxicity levels. Here "obscene", "threat", "insult" and "identity hate" are four sub-labels for "toxic" and "severe toxic" (hence may co-occur for a comment). The "toxic" comments that are not

[7] We use "generating adversarial example" and "adversarial attacks" interchangeably.

Task	Search	Transformation	N	Attack Success	Average Number	Average	Time
				Rate	of Queries	Perturbed Word %	(s)
Jigsaw-BL	delete	wordnet	-	89.29	35.35	6.68	836
		glove	5	85.61	30.77	7.04	765
		glove	20	96.58	42.37	6.71	906
		glove	50	98.27	65.05	6.55	1355
		mlm	20	93.22	40.21	6.56	1882
	unk	wordnet	-	89.59	34.87	6.66	822
		glove	5	86.04	30.55	7.06	749
		glove	20	96.75	41.87	6.68	888
		glove	50	98.38	64.17	6.55	1278
		mlm	20	93.29	39.57	6.55	1727
	wt-saliency	wordnet	-	90.27	152.05	6.6	4491
		glove	5	86.74	96.98	7.0	3646
		glove	20	96.99	288.58	6.65	7964
		glove	50	98.65	642.58	6.53	17152
		mlm	20	93.88	254.81	6.58	20093
Jigsaw-ML	delete	wordnet	-	87.29	37.05	6.83	959
		glove	5	83.81	32.21	7.46	857
		glove	20	95.75	44.27	6.9	1046
		glove	50	97.75	68.03	6.69	1665
		mlm	20	92.76	41.83	6.63	2156
	unk	wordnet	-	87.72	37.01	6.81	981
		glove	5	84.33	32.15	7.47	865
		glove	20	95.92	43.84	6.89	997
		glove	50	97.91	67.09	6.68	1530
		mlm	20	92.73	41.33	6.62	2011
	wt-saliency	wordnet	-	88.55	157.43	6.74	5331
		glove	5	84.98	100.33	7.36	3993
		glove	20	96.09	297.63	6.86	8965
		glove	50	97.89	662.08	6.65	19191
		mlm	20	93.45	262.76	6.72	22787
HTweet-MC	delete	wordnet	-	56.46	32.5	17.99	333
		glove	5	33.0	21.96	23.19	283
		glove	20	66.29	41.47	18.86	375
		glove	50	69.29	82.19	18.15	659
		mlm	20	64.98	34.34	21.34	862
	unk	wordnet	-	56.66	32.08	17.94	329
		glove	5	33.65	21.82	23.06	284
		glove	20	66.7	41.06	18.85	378
		glove	50	69.99	81.14	18.18	655
		mlm	20	65.23	33.66	21.3	851
	wt-saliency	wordnet	-	55.96	80.57	17.15	752
		glove	5	34.31	44.73	22.46	575
		glove	20	67.82	130.65	18.2	1073
		glove	50	71.31	296.44	17.71	2137
		mlm	20	65.48	104.41	21.2	2780

Table 10: Effect of word transformations on attack performance. Comparing synonym transformations only. No character transformations used. Reporting attack performance when using delete , unk , and wt-saliency greedy search. The same constraints as in Table 5 with POS (part-of-speech) match.

"obscene", "threat", "insult" and "identity hate" are assigned to either "toxic" or "severe toxic". Comments that are not assigned any of the six toxicity labels get int "non toxic".

A.4.2 Offensive Tweet.

The authors of (Davidson et al., 2017) used crowd-sourced hate speech lexicon from Hatebase.org to collect tweets containing hate speech keywords. Then they used crowd-sourcing to label these tweet samples into three categories: those containing hate speech, only offensive language, and those with neither.

A.5 Base Model Setup:

We build three base models, including {Jigsaw-BL , Jigsaw-ML , HTweet-MC } to cover three types of toxicity prediction tasks. Table 4 presents the choice of task, training/test data, transformer architecture and learning rate plus number of epochs. We use "distilbert-base-uncased" pre-trained transformers model for DistilBERT architecture. For BERT architecture, we use "GroNLP/hateBERT" pre-trained model. All texts are tokenized up to the first 128 tokens. The train batch size is 64 and we use AdamW optimizer with 50 warm-up steps and early stopping with patience 2.

The models are trained on NVIDIA T4 Tensor

Algorithm 2 Adversarial Training with AT2

```
Input: Set of all attack recipes $S_{\text{recipes}}$, number of attack recipes to exclude $N_{\text{exc}}$, set of attack recipes to use for adversarial training $S_{\text{attack}}$ (created by choosing $(|S_{\text{recipes}}| - N_{\text{exc}})$ attack recipes from the set $S_{\text{recipes}}$), number of clean epochs $N_{\text{clean}}$, number of adversarial epochs $N_{\text{adv}}$, percentage of dataset to attack $\gamma$, attack $\mathbf{A}(\theta, \mathbf{x}, \mathbf{y})$, and training data $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$
Output: model weights $\theta$
Initialize model weights $\theta$
for clean epoch$= 1, \ldots, N_{\text{clean}}$ do
    Train $\theta$ on $D$
end for
Initialize $D' \leftarrow D$
for attack recipe in $S_{\text{attack}}$ do
    $D_{\text{adv}} \leftarrow \{\}$
    $i \leftarrow 1$
    while $|D_{\text{adv}}| < \gamma * |D|$ and $i \leq |D|$ do
        $\mathbf{x}_{\text{adv}}^{(i)} \leftarrow \mathbf{A}(\theta, \mathbf{x}^{(i)}, \mathbf{y}^{(i)})$
        $D_{\text{adv}} \leftarrow D_{\text{adv}} \cup \{(\mathbf{x}_{\text{adv}}^{(i)}, \mathbf{y}^{(i)})\}$
        $i \leftarrow i + 1$
    end while
    $D' \leftarrow D' \cup D_{\text{adv}}$
end for
Randomly shuffle $D'$
for adversarial epoch$= 1, \ldots, N_{\text{adv}}$ do
    Train $\theta$ on $D'$
end for
```

Core GPUs and NVIDIA Tesla V100 GPUs with 16 GB memory, 2nd generation Intel Xeon Scalable Processors with 32GB memory and high frequency Intel Xeon Scalable Processor with 61GB memory.

A.6 Adversarial Training with Single or Multiple Attacks

Adversarial training has been a major defense strategy in most existing work for improving adversarial robustness (Madry et al., 2018). The vanilla adversarial training process involves augmenting the training data with adversarial examples generated from perturbing the training data in the input space.

Let $\mathbf{L}(\mathcal{F}, \mathbf{x}, \mathbf{y})$ represent the loss function on input text $\mathbf{x}$ and label $\mathbf{y}$. Let $\mathbf{A}(\mathcal{F}, \mathbf{x}, \mathbf{y})$ be the adversarial attack that produces adversarial example $\mathbf{x}'$. Then, vanilla adversarial training objective is as follows:

$$\begin{aligned}\arg\min_{\mathcal{F}} \mathbf{E}_{(\mathbf{x},\mathbf{y})\sim D}[&\mathbf{L}(\mathcal{F}(\mathbf{x}), \mathbf{y}) \\ &+ * \mathbf{L}(\mathcal{F}(\mathbf{x}' = \mathbf{A}(\mathcal{F}, \mathbf{x}, \mathbf{y})), \mathbf{y})]\end{aligned} \tag{4}$$

Adversarial training use both clean[8] and adversarial examples to train a model. This aims to minimize both the loss on the original training dataset and the loss on the adversarial examples.

In recent NLP studies, adversarial training (AT) is only performed to show that such training can make models more resistant to the attack it was originally trained with. This observation is not surprising. The literature has pointed out the importance of robustness against unseen attacks and it is generally recommended to use different attacks to evaluate the effectiveness of a defense strategy (Carlini et al., 2019).

Adversarial Training with Multiple Attacks (AT2): A simple strategy to revise vanilla AT is to train a model using both clean examples and adversarial examples from different attacks. This, we call Adversarial Training with Multiple (AT2), trains a target model on a combination of adversarial examples. AT2 aims to help a model become more robust not only against attacks it is trained on but also the attack recipes it has not seen before. Algorithm 2 presents our pseudo code of AT2 .

AT1 vs AT2 : In the rest of the paper, we call vanilla adversarial training as single adversarial training (AT1). In Section 3 and Section B our results show that models trained with AT2 can be more effective in protecting against unseen text adversarial attacks compare with AT1 models trained on the same attack. This could be contributed to the hypothesis that an unseen attack may share similar underlying attributes and patterns with the attack ensemble that the model is trained on.

Selecting what attacks to use in AT2 is important. Part of the reason we select to adapt the five popular recipes to ToxicTraps Extend in Table 14 is because these attacks cover a good range of popular transformations and constraints. Table 14 includes three word based attacks, two character based attack, plus TT-TBug is a character and word level combination attack. In our experiments, we simulated potential AT2 use cases by leave-one attack out as "unseen" and train AT2 models using the rest. For instance, when a target model never uses examples from TT-TFool in training, the AT2 trained

[8] *Clean* examples refer to the original training examples.

model may have already known certain information on similar word transformations since similar transformations have been used by other attacks in the ensemble.

Base Model	Search	POS	Toxic Examples Attacked (%)	Attack Success Rate	Average Number of Queries	Average Perturbed Word	Time (s)
Jigsaw-BL	gradient		59.76	98.74	34.68	7.58	849
	gradient	x		98.72	26.78	7.06	883
	delete			99.42	55.38	7.38	915
	delete	x		99.21	48.03	6.8	1070
	unk	x		99.27	47.62	6.76	939
	wt-saliency	x		99.19	407.43	6.71	10653
	genetic	x		92.67	846.41	8.73	21504
	beam	x		99.68	658.55	6.95	16043
Jigsaw-ML	gradient		65.46	97.62	38.45	8.36	1066
	gradient	x		97.72	29.78	7.56	1151
	delete			98.71	57.6	7.54	1112
	delete	x		98.63	49.93	6.99	1168
	unk	x		98.75	49.38	6.96	1141
	wt-saliency	x		98.51	419.58	6.91	12849
	genetic	x		88.91	876.81	8.82	26137
	beam	x		99.54	756.05	7.19	21290
HTweet-MC	gradient		96.62	67.16	63.51	24.13	554
	gradient	x		67.56	49.67	24.08	581
	delete			71.46	58.78	18.96	412
	delete	x		71.46	48.17	19.37	425
	unk	x		72.23	48.04	19.43	425
	wt-saliency	x		74.71	178.91	18.81	1467
	genetic	x		80.49	1025.66	21.86	6560
	beam	x		90.07	442.18	18.76	2938

Table 11: Effect of with or without part-of-speech constraints when combining with different search strategies on attack performance. The Search column identifies type of search method used. The POS column identifies if part-of-speech matching constraint is used.

Task	Training	Search	Attack Success Rate	Average Number of Queries	Average Perturbed Word	Time (s)
Jigsaw-BL	No AT	unk	99.27	47.62	6.76	939
	AT1-unk	unk	60.28	103.54	13.03	3546
Jigsaw-ML	No AT	unk	98.75	49.38	6.96	1141
	AT1-unk	unk	71.7	91.08	11.84	3106
HTweet-MC	No AT	delete	71.46	48.17	19.37	425
	No AT	unk	72.23	48.04	19.43	425
	AT1-delete	delete	21.97	69.68	28.83	598
	AT1-unk	delete	11.17	74.71	33.07	641
	AT2	delete	8.28	75.71	27.89	672
	AT2	unk	6.08	77.91	33.24	681

Table 12: Effect of adversarial training on attack performance on three tasks. When attacking, `ToxicTrap` uses the `glove` with $N = 20$ plus character transformations; constraints with POS; and search with two different greedy methods.

Task	Training	AUC	AP	F1	Recall
Jigsaw-BL	No AT	0.971	0.749	0.823	0.794
	AT1-unk	0.971	0.733	0.823	0.811
Jigsaw-ML	No AT	0.984	0.636	0.587	0.515
	AT1-unk	0.984	0.633	0.594	0.535
HTweet-MC	No AT	0.935	0.786	0.730	0.710
	AT1-delete	0.936	0.792	0.740	0.719
	AT1-unk	0.938	0.785	0.738	0.723
	AT2	0.932	0.778	0.685	0.641

Table 13: Effect of adversarial training on model performance. Macro-average metrics are reported.

B Extra on ToxicTrap Extensions

B.1 A Suite of Attack Recipes to Extend ToxicTrap

Morris et al. (2020b) splits each text adversarial attack into four parts: goal function, transformation, search strategy and constraints. With this modular design, new NLP attacks frequently consist of just or two new components and often re-use remaining components from past work. We follow this design process, using our newly proposed ToxicTrap goal functions to pair with popular choices of other three components from the literature to get a set of ToxicTraps Extend recipes.

We select five popular attack recipes from the literature including DeepWordBug , TextBugger , A2T , PWWS and TextFooler that were popular recipes proposed to attack general language classifiers. By swapping these recipes' goal function with the three goal function we propose in Equation (2) and Equation (3), we construct 15 new ToxicTraps Extend attack recipes as shown in Table 14. This table categorizes different ToxicTraps Extend attack recipes, based on their goal functions, constraints, transformations and search methods.

TT-Dbug and TT-TBug are adapted from two SOTA attack recipes DeepWordBug (Gao et al., 2018) and TextBugger (Li et al., 2019). They generate ToxicTraps Extend via character manipulations. These attacks perform character insertion, deletion, neighboring swap and replacements to change a word into one that a target toxicity detection model doesn't recognize. These character changes are designed to generate character sequences that a human reader could easily correct into those original words. Language semantics are preserved since human readers can correct the misspellings.

We then select three other popular attacks A2T , PWWS and TextFooler (Yoo and Qi, 2021; Ren et al., 2019; Jin et al., 2019) to ToxicTraps Extend TT-A2T , TT-Pwws and TT-TFool attacks. These attacks generate adversarial examples via replacing words from the input with synonyms. These attacks aim to create examples that preserve semantics, grammaticality, and non-suspicion. They vary regarding the word transformation strategies they use (see Table 14 for details).

All ToxicTraps Extend attack recipes use greedy based word importance ranking (Greedy-WIR) strategy to search and determine what words to manipulate (with character changes or with synonym replacement).

Lastly, these ToxicTraps Extend recipes also have difference in what languages constraints they employ to limit the transformations, for instance, TT-A2T puts limit on the number of words to perturb. TT-TBug uses universal sentence encoding (USE) similarity as a constraint.

Table 15 presents a few generated ToxicTraps Extend examples from attacking base models using five ToxicTraps Extend attacks.

Attack Recipe	Constraints	Transformation	Search Method
TT-A2T (revised from (Yoo and Qi, 2021))	Sentence-transformers/all-MiniLM-L6-v2 sentence encoding cosine similarity > 0.9 †, Part-of-speech match, Ratio of number of words modified < 0.1	Word Synonym Replacement. Swap words with their 20 nearest neighbors in the counter-fitted GLOVE word embedding space or optionally with those predicted by BERT MLM model.	Greedy-WIR (gradient-based)
TT-TFool (revised from (Jin et al., 2019)	Word embedding cosine similarity > 0.5, Part-of-speech match, USE sentence encoding angular similarity > 0.84	Word Synonym Replacement. Swap words with their 50 nearest neighbors in the counter-fitted GLOVE word embedding space.	Greedy-WIR (deletion-based)
TT-Pwws (revised from (Ren et al., 2019))	No special constraints	Word Synonym Replacement. Swap words with synonyms from WordNet.	Greedy-WIR (saliency)
TT-Dbug (revised from (Gao et al., 2018))	Levenshtein edit distance < 30	{Random Character Insertion, Random Character Deletion, Random Character Swap, Random Character Replacement}*	Greedy-WIR (deletion-based)
TT-TBug (revised from (Li et al., 2019))	USE sentence encoding cosine similarity > 0.8	{Random Character Insertion, Random Character Deletion, Neighboring Character Swap, Character Substitution by Homoglyph ‡, Word Synonym Replacement with 5 nearest neighbors in the counter-fitted GLOVE word embedding space}*	Greedy-WIR (deletion-based)

Table 14: ToxicTraps Extend Attack recipes categorized regarding: search method, transformation, goal function, constraints. All attack recipes include an additional constraint which disallows the replacement of stopwords. Greedy search with Word Importance Ranking (Greedy-WIR).

† *indicates new features or enhancements we added to extend the capability of the original attack recipes proposed in the original papers*

* *indicates a combination of multiple transformations.* ‡ *substitute a character with visually similar ones or those adjacent on keyboard. For example, substitute "1" with "l".*

Seed Text	`TT-A2T`	`TT-TFool`	`TT-Pwws`	`TT-Dbug`	`TT-TBug`
My neighbors hate me cuz i smoke **weed** and blast the most ghetto music f...	My neighbors hate me cuz i smoke **weeds** and blast the most ghetto music fr...	My neighbors hate me cuz i smoke **pot** and blast the most ghetto music from...	My neighbors hate me cuz i smoke **smoke** and blast the most ghetto music fr...	My neighbors hate me cuz i smoke **wee** and blast the most ghetto music from...	My neighbors hate me cuz i smoke **wed** and blast the most ghetto music fro...
thank god FOB didnt come back doing some **shitty** arena tour. #iwaited5ye...	thank god FOB didnt come back doing some **crappy** arena tour. #iwaited5year...	thank god FOB didnt come back doing some **foul** arena tour. #iwaited5yearsf...	thank god FOB didnt come back doing some **stinky** arena tour. #iwaited5year...	thank god FOB didnt come back doing some **sitty** arena tour. #iwaited5years...	thank god FOB didnt come back doing some **crappy** arena tour. #iwaited5year...
Fluffy is a **retard** homeboy is calling a baby rattlesnake friend and kis...	Fluffy is a **brainless** homeboy is calling a baby rattlesnake friend and ki...	Fluffy is a **moron** homeboy is calling a baby rattlesnake friend and kissin...	Fluffy is a **changeling** homeboy is calling a baby rattlesnake friend and k...	Fluffy is a **Getard** homeboy is calling a baby rattlesnake friend and kissi...	Fluffy is a **retarded** homeboy is calling a baby rattlesnake friend and kis...
Typically hateful, anti-Christian, mentally ill and ugly **dyke** trash pig ...	Typically hateful, anti-Christian, mentally ill and ugly **levee** trash pig ...	Typically hateful, anti-Christian, mentally ill and ugly **embankment** trash...	Typically hateful, anti-Christian, mentally ill and ugly **dike** trash pig c...	Typically hateful, anti-Christian, mentally ill and ugly **dmyke** trash pig ...	Typically hateful, anti-Christian, mentally ill and ugly **dike** trash pig c...
How ghetto **girls** pronounce there names... I could cry right now oh my g...	How ghetto **females** pronounce there names... I could cry right now oh my g...	How ghetto **females** pronounce there names... I could cry right now oh my g...	How ghetto **missy** pronounce there names... I could cry right now oh my god...	How ghetto **girsl** pronounce there names... I could cry right now oh my god...	How ghetto **gilrs** pronounce there names... I could cry right now oh my god...
More **queer** lady Templars. That's my point.	More **homo** lady Templars. That's my point.	More **fairy** lady Templars. That's my point.	More **curious** lady Templars. That's my point.	More **qIeer** lady Templars. That's my point.	More **homo** lady Templars. That's my point.

Table 15: Selected Toxic Adversarial Examples. We show adversarial examples generated by attacking base model `HTweet-MC` . To conserve space, we only show results from Offensive Tweet that contain much shorter messages than Jigsaw.