# Adversarial Robustness in Deep Neural Networks for Spam Email Detection

**Le Hong Trieu (A0057066A), Wang Shining (A0177386M),**
**Wang Tianchi (A0227468U), Zhu Sisi (A0130509N)**
School of Computing
National University of Singapore
{e0945456, e0253746, e0656981, e0924674}@u.nus.edu

## Abstract

Spam emails pose significant risks in today's digital landscape, challenging advanced filtering systems to both individuals and organizations worldwide. Adversarial attacks aim to create emails that evade detection while maintaining human readability. However, existing methods are often tailored for continuous data like images, leaving a gap for discrete data like emails. Common attacks include using word replacement strategies that leverage word embeddings, counter-fitted embeddings, thesauruses, or a BERT mask language model. This paper proposes a unified framework supporting both white-box and black-box attacks using large language models such as BERT or GPT to generate high-quality adversarial spam samples with minimal computational cost. This approach is effective in deceiving various deep neural network models, including multilayer perceptron, LSTM, ALBERT, DistilBERT, and BERT. Additionally, we explore defense strategies, including adversarial training and regularization techniques, effectively restoring spam emails prediction. The implementation code and generated adversarial spam emails are available on our GitHub [1].

## 1 Introduction

Spam emails flood inboxes, posing significant risks globally. By 2023, they comprised 45.6% of total email traffic[14], with a $20 billion annual economic impact[8]. Beyond financial losses, spam undermines productivity and social security[10]. Adversarial attacks on spam filters worsen this threat, compromising filter integrity.

In this project, we implemented a unified framework incorporating gradient-based white-box attacks and black-box attacks along with large language models, such as BERT and the GPT-4 API, to replace important tokens in various neural network models, including MLP, LSTM, ALBERT, DistilBERT, and BERT. The attacks successfully deceived original classifier on spam email detection. For defense, we adopted adversarial training and regularization techniques and effectively restored spam email prediction. We also contributed a dataset of adversarial spam emails that exhibit high quantitative and qualitative similarity scores, generated via both white-box and black-box attacks across all targeted models.

With our method, we aim to develop a fortified and robust email filter for spam detection. The project outcome can be used to mitigate social, economic and cybersecurity risk of spam, and create a safer online environment.

[1] Our GitHub: https://github.com/trieu-le-arner/adversarial_attacks_spam_filters

Preprint. Under review.

## 2 Related Works

### 2.1 Adversarial Robustness

Deep neural networks are susceptible to adversarial attacks due to their high-dimensional nature and steep loss gradients [9]. To enhance robustness, training involves solving a min-max optimization problem described as: $\min_\theta \frac{1}{|S|} \sum_{(x,y)\in S} \max_{\|\delta\|\leq\epsilon} l(h_\theta(x+\delta), y)$. This approach aims to minimize the maximum potential loss from adversarial perturbations. Optimization employs gradient descent, guided by Danskin's Theorem [18], which involves computing gradients at points of maximum loss. Empirically, adversarial attacks successfully generate samples through local search methods, where perturbation vectors $\delta$ satisfy the projection constraint $\|\delta\| \leq \epsilon$ to preserve original semantics [9].

### 2.2 Adversarial Attacks and Training for Discrete Text Inputs

Much research on adversarial attacks has focused on continuous images, while attacking discrete data like text is more challenging due to the lack of direct gradient signals back into the discrete token space. In transparent models, gradient-based methods such as the Fast Gradient Sign Method (FGSM) [4] and Projected Gradient Descent (PGD) [9] can modify continuous word embeddings and then identify nearby candidate replacements. Alternatively, token manipulation methods like Semantically Equivalent Adversaries [15] and Easy Data Augmentation [17] alter text through minimal semantic changes or random text modifications. Attacks like TextFooler [7] and BERT-Attack [11] further refine this by targeting specific words and generating substitutions using methods like greedy search or the BERT model.

### 2.3 Adversarial Attacks against Spam Filters

Token manipulation, particularly adding and substituting words, effectively breaches traditional spam filters like the Naive Bayes algorithm [10]. Similarly, neural network-based spam filters, despite their higher accuracy, are also vulnerable to adversarial methods like Feature Perturbation [2]. This technique employs Projected Gradient Descent (PGD) to craft spam emails by inserting specific "magic words" to enhance the emails' chance of bypassing filters.

## 3 Our Approach

### 3.1 Baseline Random Text Perturbation Attack and Defense

Table 1: Baseline model (LSTM) attacker

| Attack | Level |
|---|---|
| Sentence Shuffling | Sentence |
| Synonyms Replacement | Token |
| Homoglyphs Replacement | Character |
| Keyboard Typo Simulation | Character |
| Special Character Insertion | Character |

To assess the impact of adversarial attacks, we experimented with random black-box text perturbations as our baseline. We trained an LSTM-based spam filter and implemented a text perturbation attacker as outlined in Table 1. We defined 10 attack configurations with various perturbation settings as in Table 10. With 10 attack rates ranging from 0.1 to 1.0, reflecting each token's perturbation probability, we produced 100 datasets with varying attack intensities using spam emails.

### 3.2 The Unified Attack Framework

We developed a unified framework for conducting both white-box and black-box attacks utilizing large language models like BERT and GPT-4 API to translate the outcomes into discrete word spaces. The framework includes three key components: computing importance scores to select tokens in spam emails, identifying top-$k$ candidate replacements for each important token, and employing a greedy search to optimize word substitution.

Our goal with adversarial attacks is to misclassify spam emails as non-spam, aligning with typical objectives of malicious actors. Conversely, inducing classifiers to erroneously label non-spam emails as spam is not our focus.

#### A. Compute Importance Scores for White-box and Black-box Attacks

**White-box** gradient-based attacks prioritize token importance by perturbation deltas. For classifiers using TF-IDF vectors, more negative delta values highlight candidate replacements, whereas clas-

Table 2: Gradient-based attacks for perturbation vector generation

| Attack Acronym | Attack | Perturbation Delta Generation |
|---|---|---|
| FGSM | The Fast Gradient Sign Method | $l = \text{BCELoss}()$<br>$\delta = \epsilon \times \text{sign}(\nabla_\delta l(h_\theta(x+\delta), y))$ |
| PGD-$L_\infty$ | Projected Gradient Descent with $L_\infty$ norm | $l = \text{BCELoss}()$<br>Repeat:<br>$\delta_{t+1} = \delta_t + \alpha \times \text{sign}(\nabla_\delta l(h_\theta(x+\delta), y))$<br>$\delta_{t+1} = \delta_{t+1}.\text{clamp}(-\epsilon, \epsilon)$ |
| PGD-$L_2$ | Projected Gradient Descent with $L_2$ norm | $l = \text{BCELoss}()$<br>Repeat:<br>$\delta_{t+1} = \delta_t + \alpha \times \frac{\nabla_\delta l(h_\theta(x+\delta), y)}{\|\nabla_\delta l(h_\theta(x+\delta), y)\|_2}$<br>$\delta_{t+1} = \delta_{t+1} \times \frac{\epsilon}{\|\delta_{t+1}\|_2.\text{clamp(min=}\epsilon)}$ |
| Targeted PGD-$L_\infty$-1 | Targeted Attack 1 using Projected Gradient Descent with $L_\infty$ norm | $l = h_\theta(x+\delta)_{y=0} - h_\theta(x+\delta)_{y=y_{true}}$<br>Repeat:<br>$\delta_{t+1} = \delta_t + \alpha \times \text{sign}(\nabla_\delta l(h_\theta(x+\delta), y))$<br>$\delta_{t+1} = \delta_{t+1}.\text{clamp}(-\epsilon, \epsilon)$ |
| Targeted PGD-$L_\infty$-2 | Targeted Attack 2 using Projected Gradient Descent with $L_\infty$ norm (Targeted PGD-$L_\infty$-2) | $l = h_\theta(x+\delta)_{y=0} - h_\theta(x+\delta)_{y=1}$<br>Repeat:<br>$\delta_{t+1} = \delta_t + \alpha \times \text{sign}(\nabla_\delta l(h_\theta(x+\delta), y))$<br>$\delta_{t+1} = \delta_{t+1}.\text{clamp}(-\epsilon, \epsilon)$ |

sifiers using embedding vectors focus on tokens with higher mean absolute delta values. Attention masks help exclude pad tokens from this ranking.

The framework began with FGSM, targeting extreme corners of the $\epsilon$-hypercube. We tested two PGD variants differentiated by their update and clamping strategies, one confined to a hypercube ($L_\infty$) and the other to a hypersphere ($L_2$). These untargeted attacks aim to misclassify spam as non-spam and vice versa.

Targeted $L_\infty$ PGD aims to specifically misclassify spam as non-spam using two loss function strategies. The first minimizes the true label's logit while maximizing the non-spam logit, and the second maximizes the non-spam logit while minimizing the spam logit. These approaches are different when the true label is non-spam; the first variant maintains a zero loss, whereas the second variant applies a penalty to prevent misclassification of non-spam emails as spam.

The mathematical representations of all conducted gradient-based attacks are summarized in Table 2.

In **black-box** attacks, where the adversary only has access to the output logits of a targeted classifier, we adopt a strategy inspired by BERT-Attack [11]. Tokens in a spam email are replaced with mask tokens, creating $n$ masked emails for the $n$ tokens. These are processed by the targeted BERT classifier in batches to determine the importance of each token based on the difference in logits between the original and masked emails. A larger difference signifies a token's higher importance for potential misclassification. The importance score for the $i$th token is calculated as: $I_{w_i} = o_y(S) - o_y(S_{\setminus w_i})$ where $S$ is the input email and $o_y(S)$ is the output logit from BERT. $S_{\setminus w_i}$ represents the email with the $i$th token masked.

The logic for ranking tokens by importance scores for substitution is detailed in Pseudocode 1.

### B. Identify Top-$k$ Candidate Replacements by a Large Language Model

We developed two strategies for generating top-$k$ candidate replacements for important tokens using large language models like BERT or GPT. The first strategy utilizes the masking feature or next-word prediction of these models; each important token is masked and queried for $k$ potential replacements, preserving the semantics of the original spam email. This process requires multiple queries, one for each important token, leading to substantial computational costs. For example, generating replacements for 200 important words across 25 emails costs about $80, equivalent to processing 8 million tokens by OpenAI's GPT-4 API.

The second strategy minimizes computational expenses by utilizing BERT's output structure, formatted as [batch, sequence length, vocabulary size]. Vocabulary size reflects the probability of each token in the vocabulary being a suitable replacement. With a single query, BERT identifies the top-$k$ potential replacements for every important token, reducing costs significantly. This method can also be adapted for GPT-4 API calls, generating all top-$k$ replacements for each important word in one request. Costs are reduced by a factor of $n$, the number of selected important words per spam email. Each important token is concatenated with its adjacent tokens to reconstruct complete words prior to submission through the GPT-4 API.

The structured prompt used for GPT-4 API calls is as follows:

```
For each word in the word list, find the top {k} synonyms (words or phrases) that meet these two conditions:
- Replacing occurrences of the word in the text with them does not alter the meaning and grammar of the text.
- Sort the top {k} synonyms by decreasing relevance score within the text's context. The higher the score, the
      more relevant a replacement word is to the text.

Format as "word: replacement words" in a single line. Wrap all lines in between a single pair "<<<" and ">>>".

Text:
{spam_sample}

Word list:
{important_words}
```

### C. Word or Phrase Substitution by a Greedy Search

We devised a simple greedy search algorithm, inspired by TextAttack [13] and BERT-Attack [11]. The algorithm iterates through each important token ranked in Step 3.2.A. For each token, it further iterates through each candidate substitution identified in Step 3.2.B to iteratively draft an adversarial spam email. Upon creating each adversarial email, a query is submitted to the targeted classifier. If the classifier identifies the email as non-spam, the substitution attack is deemed successful, and the process stops. If the email is still classified as spam, the algorithm continues to the next substitution, recording the most effective substitution thus far that increases the likelihood of being classified as non-spam.

### 3.3 Adversarial Defenses

We implemented two defensive approaches against the aforementioned attacks. First, we incorporated dynamically generated adversarial samples from gradient-based attacks and text augmentation using text perturbation heuristics, such as synonyms, homoglyphs, keyboard typos, and special character insertion during training. The loss function [4] is mathematically expressed as follows:

$$L = \frac{1}{(m-k)+\lambda k}\left(\sum_{i\in\text{ORIGINAL}} L(\theta, x_i, y_i^{true}) + \lambda \sum_{i\in\text{ADVERSARIAL}} L\left(\theta, \tilde{x}_i, y_i^{true}\right)\right),$$

where $m$ represents the batch size, $k$ denotes the number of adversarial samples that replace the original samples in the batch, and $\lambda$ controls the relative weight of the adversarial samples in the loss function.

Second, we introduced a method to prevent overfitting by utilizing Flooding-X [12], which is reflected in the modified loss function: $\tilde{L} = |L(\theta) - b| + b$. This technique aids in regularization to enhance model generalization and prevents the training loss reduction beyond a certain threshold.

## 4 Experiments

### 4.1 Datasets and Pre-processing

| Dataset | Non-spam | Spam |
|---|---|---|
| [16] Assassin | 6954 | 3795 |
| [3] Enron | 16545 | 17171 |
| [5] Ling | 2412 | 481 |
| Pre-processed | 21154 | 15714 |

For records stored in multiple parts, we merged them into a single continuous text. Using BeautifulSoup, we extracted plain text from HTML-formatted emails. We removed new lines, tabs, and rare special characters. Non-English emails were excluded, and only those with more than 50 characters were retained. Duplicates across all three datasets were eliminated. The combined dataset was then divided into train, test, and validation sets for model building and evaluation.

Table 3: Evaluation metrics

| Metric | Description |
|---|---|
| Original Accuracy | The accuracy of a targeted model before it is attacked. |
| Attacked Accuracy | The accuracy of a targeted model after it has been attacked. |
| Robust Accuracy | The accuracy of a targeted model after undergoing adversarial defending. |
| Similarity Score | The measurement of semantic consistency between original spam emails and their adversarial counterparts using Universal Sentence Encoder [1]. |
| Change Count | The number of words substituted to generate adversarial spam emails, measuring the effectiveness of an attack. |
| Query Count | The number of times a targeted model is queried to compute important scores and prediction feedback, measuring the efficiency of an attack. |

### 4.2 Metrics

In evaluating aforementioned attacks, including gradient-based white-box and logit-based black-box attacks, targeted models like MLP, LSTM, BERT, DistilBERT, and ALBERT, alongside different feature representations such as TF-IDF and embedding vectors, and the utilization of various large language models like BERT and GPT-4 API for mapping into discrete word space, we employed the evaluation metrics described in Table 3.

### 4.3 Results

**A. Evaluation of Baseline Random Text Perturbation Attack**

Our initial model accuracy reached 0.98 as annotated with green dot line in the results (Figure 1). It shows that the model accuracy declined as the attack rate approached 0.5. As attack rate approached to 1.0, accuracy dropped to 0.5 for attack *Keyboard Typo Simulation* and *Special Character Insertion*. However, *Synonyms Replacement* and *Homoglyphs Replacement* showed minimal impact to the model accuracy, indicating our baseline model's resilience. As expected, the text similarity decreased with higher attack rates (see Figure 2).

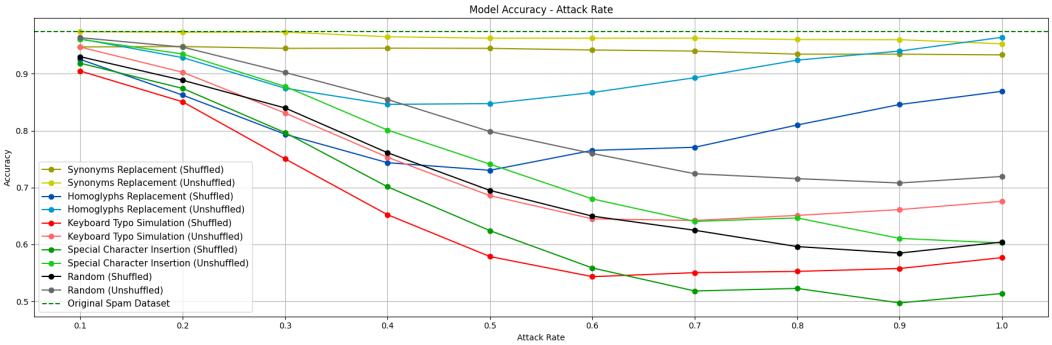


Figure 1: Attacked baseline model (LSTM) classification accuracy vs attack rate

**B. Evaluation of Unified Framework Adversarial Attacks**

As indicated in Table 4, all gradient-based white-box attack methods with the GPT-4 API successfully deceived both MLP and ALBERT targeted models, leading them to misclassify spam emails. DistilBERT is robust against gradient-based attacks, as they were not particularly effective in compromising its performance compared to other models under the same attacks.

The Fast Gradient Sign Method (FGSM) is not effective for computing the importance score of each token in a spam email because it maps each perturbed token to an extreme corner of their hypercube. Consequently, it fails to show the relative difference in terms of importance between tokens.

We also experimented with ensemble attacks, where the two strongest attacks for each targeted model were combined to identify important tokens for substitution. For MLP, the ensemble attack PGD-$L_\infty$+PGD-$L_2$ performed sub-optimally, yielding the highest average query count among all attacks.

Table 4: Results of attacks on various targeted neural models using gradient-based white-box and logit-based black-box techniques, and mapping into discrete word space by large language models such as BERT and GPT

| **Targeted Model** | **LLM** | **Avg Original Acc** | **Attack** | **Avg Attacked Acc** | **Defend** | **Avg Robust Acc** | **Avg Similarity Score** | **Avg Change Count** | **Avg Query Count** |
|---|---|---|---|---|---|---|---|---|---|
| **LSTM** | - | 0.96 | Text perturbation | 0.66 | Adv training | 0.78 | 0.351 | - | 0 |
| **MLP + TF-IDF** | GPT-4 API | 1.0 | PGD-$L_\infty$ | **0.59** | Adv training | 0.93 | 0.91 | 8.61 | 174.35 |
| | | | PGD-$L_2$ | 0.61 | | 0.96 | 0.91 | 8.36 | 176.03 |
| | | | Targeted PGD-$L_\infty$-1 | 0.63 | | 0.97 | 0.90 | 8.84 | 178.61 |
| | | | Targeted PGD-$L_\infty$-2 | 0.61 | | 0.99 | 0.90 | 8.64 | 167.89 |
| | | | PGD-$L_\infty$+PGD-$L_2$ | 0.87 | | - | 0.88 | 4.26 | 199.06 |
| **ALBERT** | GPT-4 API | 0.97 | PGD-$L_\infty$ | 0.69 | Adv training | 0.98 | 0.86 | 23.69 | 101.50 |
| | | | PGD-$L_2$ | 0.63 | | 0.98 | 0.85 | 28.16 | 112.05 |
| | | | Targeted PGD-$L_\infty$-1 | 0.68 | | 1.00 | 0.86 | 25.13 | 99.00 |
| | | | Targeted PGD-$L_\infty$-2 | **0.58** | | 0.98 | 0.85 | 27.52 | 107.79 |
| | | | Targeted PGD-$L_\infty$-2+PGD-$L_2$ | **0.58** | | - | 0.85 | 27.40 | 104.93 |
| **DistilBERT** | GPT-4 API | 0.99 | PGD-$L_\infty$ | 0.92 | Adv training | 0.99 | 0.86 | 26.49 | 93.41 |
| | | | PGD-$L_2$ | 0.90 | | 0.99 | 0.85 | 26.97 | 92.88 |
| | | | Targeted PGD-$L_\infty$-1 | **0.85** | | 0.99 | 0.85 | 28.76 | 98.46 |
| | | | Targeted PGD-$L_\infty$-2 | 0.86 | | 0.99 | 0.84 | 29.51 | 99.33 |
| | | | Targeted PGD-$L_\infty$-2+PGD-$L_2$ | **0.85** | | - | 0.83 | 29.46 | 101.75 |
| **BERT** | BERT | 0.96 | Black-box based on logits | **0.58** | Flooding-X | 0.97 | 0.75 | 5.61 | 851.52 |

Table 5: Results of adversarial attacks from other studies

| **Targeted Model** | **Avg Original Acc** | **Attack** | **Avg Attacked Acc** | **Defend** | **Avg Change Count** |
|---|---|---|---|---|---|
| **SVM + TF-IDF** [2] | 0.96 | PGD-$L_2$ | 0.51 (Only Ling dataset) | Adv training | 29.0 |
| **ECLF + Word2vec** [2] | 0.96 (Weighted avg) | Black-box with magic word insertions | 0.60 (Weighted avg) | Adv training | - |

This indicates inefficiency in computing important scores, resulting in a higher average attacked accuracy of $0.87$. However, for ALBERT, the ensemble attack Targeted PGD-$L_\infty$-2+PGD-$L_2$ was the most successful. It matched the lowest attacked accuracy of Targeted PGD-$L_\infty$-2 but with a lower average query count, suggesting that the ensemble method more effectively identified important words.

Gradient-based white-box attacks on ALBERT demonstrated a lower average query count, indicating a more effective discovery of substitutions. Notably, for MLP, we queried the GPT-4 API for the top-5 potential replacements, while for ALBERT, we initially considered only the top-2 candidates. Expanding the query to the top-4 potential candidates for ALBERT led to a significant decrease in attacked accuracy to $0.33$, as shown in Table 6. This indicates that more complex models like ALBERT are more sensitive to gradient-based attack in terms of most-important-word replacement, consistent with findings from the prior study [6].

Conversely, attacks on ALBERT involved substituting three times more words than on MLP, resulting in adversarial spam emails with much lower average similarity scores. These substitutions effectively attacked ALBERT but resulted in only a minimal reduction per substitution in the likelihood of spam samples being classified as spam, compared to MLP. For MLP, while many substitutions did not significantly impact misclassification, a few had a profound effect, suggesting ALBERT's relative robustness against these attacks.

Our logit-based black-box attacks on the BERT model significantly reduced the prediction accuracy for spam email classification from $0.96$ to $0.58$. This method averaged only $5.61$ word substitutions per attack, fewer than those made by the GPT-4 API. The similarity score between the original and adversarial emails was the lowest at $0.75$, breaking down to $0.91$ for successful attacks and $0.64$ for unsuccessful ones. This pattern reflects the method's approach of continuous token substitution until the attack succeeds, with unsuccessful attempts involving more substitutions and thus lower similarity scores. The average query count was $851$, higher than in white-box attacks, due to the extensive ranking of token importance and the greedy search for optimal substitutes. This makes the attack more expensive and resource-intensive compared to the less demanding gradient-based white-box attacks used with the GPT-4 API for this email dataset.

### C. Comparison Against Adversarial Attacks from Other Studies

In comparison to the white-box PGD-$L_2$ attack on TF-IDF targeting only the Ling dataset as described in another study [2], our approach yielded an attacked accuracy of $0.44$, which is better than their $0.51$, as shown in Table 5. Their methodology involved identifying words with significant variations due to PGD-$L_2$ perturbations that intersect with 'ham' words—those found in non-spam emails but absent in spam. These words were then inserted into perturbed spam emails, resulting in an average of $29.0$ words being added, as opposed to $11.3$ words replaced in our method. Additionally, their strategy did not account for the semantic coherence of the inserted words within the original spam emails' context, which likely explains the absence of similarity scores in their report.

For the black-box attack comparison, as captured in Table 5, our method outperformed the approach detailed in the same referenced research paper, achieving an attacked accuracy of $0.58$ compared to the $0.60$ recorded for the ensemble classifier (ECLF) with word2vec embedding, as reported in the study and calculated as a weighted average across three datasets.

| k | Attacked Acc | Sim Score |
|---|---|---|
| 2 | 0.580 | 0.847774 |
| 3 | 0.430 | 0.827934 |
| 4 | 0.330 | 0.837014 |

Table 6: Impact of Top-k Candidate Size

| Ranking Size | Attacked Acc |
|---|---|
| 100 | 0.420 |
| 150 | 0.340 |
| 200 | 0.330 |

Table 7: Impact of Important Token Size

**D. Evaluation of Adversarial Defense Models**

The baseline model was retrained using adversarial samples with an augmentation rate of 0.5, reflecting each spam email's probability of being augmented with random attacks. By re-executing the attack experiments, our results (Figure 3) show that the retrained model successfully defended majority of perturbed samples and the accuracy reached $0.98$. It is also shown that the retrained model remains effective in classifying the original dataset and is not overfitting perturbed text (see Table 12). The effectiveness of adversarial training at attack rate of 0.8 is detailed in Table 13.

Adversarial defense models overall performed well on the spam-only dataset, which was adversarially pre-generated using random text perturbation or a gradient-based white-box attack in conjunction with GPT-4 for discrete word mapping. These models were trained with both adversarial spam and non-spam samples created in continuous space by the same attack, as shown in the 'Robust Accuracy' column of Table 4.

Flooding-X [12], an adversarial defense technique, was applied to the BERT model to prevent overfitting during the fine-tuning stage. It successfully defended against the BERT-attack and achieved $0.97$ accuracy in predicting adversarial spam emails. The technique effectively boosted robustness accuracy at an earlier stage while being efficient, as it required no adversarial samples, compared to the more expensive generation of adversarial samples during adversarial training.

**E. Adversarial Defense Models against Other Attacks**

All gradient-based defense models are robust against all other gradient-based attacks as shown in Table 8, reaching accuracy above 0.98. Flooding-X defense technique is also robust against gradient-based attack on ALBERT and text perturbation on LSTM.

### 4.4 Ablations

**A. Impact of Top-k Candidate Size**

As shown in Table 6, the effectiveness of the attack Targeted PGD-$L_\infty$-2 on ALBERT increases with the size of candidate $k$; however, the semantics of the generated adversarial samples are less preserved.

**B. Impact of Important Token Size**

Table 7 shows that increasing the number of important tokens enhances the effectiveness of the Targeted PGD-$L_\infty$-2 attack on ALBERT. Conversely, when using a random token importance ranking, the attack is much less effective, with an attacked accuracy is high at $0.97$.

**C. Impact of Adversarial Sample Size during Adversarial Training**

With a small adversarial sample size $k$ during adversarial training against gradient-based attacks, such as $0.05$ or $0.1$, the model learned to become robust against datasets pre-generated by all attacks and GPT-4 prompts. However, with a larger value like $k = 0.5$, the robust accuracy after adversarial training against PGD-$L_\infty$ plummeted to $0.0$ when defending against all other gradient-based attacks, even with a small adversarial weight $\lambda = 0.1$. Despite this, learning from adversarial samples generated by other gradient-based attacks remains effective against all such attacks when $k = 0.5$.

### 4.5 Qualitative Evaluation

Table 9 shows some generated adversarial examples. For additional examples, please visit our GitHub repository.

## 5 Conclusion

In terms of attacks, using complex methods to determine the importance of tokens leads to more effective results. By focusing on the detailed significance of each token, our attacks greatly challenge the strength of our original classifiers.

On the defense side, we focused on two main strategies: adversarial training and Flooding-X. Adversarial training has effectively enhanced our model's resilience against attacks, though it is computationally demanding. In contrast, the Flooding-X technique offers a compelling alternative by increasing resilience without needing adversarial samples. This method shows great promise in improving the effectiveness and robustness of our defense approach.

Another observation is that gradient-based attacks combined with GPT-4 prompting vary in effectiveness depending on the model complexity. In simpler models like MLP, finding effective substitutions is challenging, but successful substitutions significantly reduce spam probability. Conversely, in more complex models like ALBERT, identifying suitable replacements is easier, but these substitutions result in smaller reductions in spam probability.

## 6 Contributions

Each member in our group has equal contributions to this project.

## References


[1] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[2] Q. Cheng, A. Xu, X. Li, and L. Ding. Adversarial email generation against spam detection models through feature perturbation. In *2022 IEEE International Conference on Assured Autonomy (ICAA)*, pages 83–92. IEEE, 2022.

[3] H. Face and SetFit. Enron spam dataset. `https://huggingface.co/datasets/SetFit/enron_spam`, 2022. Accessed: 2024-03-12.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] M. Gu. Lingspam dataset. `https://www.kaggle.com/datasets/mandygu/lingspam-dataset`, 2020. Accessed: 2024-03-12.

[6] C. Guo, A. Sablayrolles, H. Jégou, and D. Kiela. Gradient-based adversarial attacks against text transformers. *arXiv preprint arXiv:2104.13733*, 2021.

[7] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits. Is bert really robust? natural language attack on text classification and entailment. corr abs/1907.11932 (2019). *arXiv preprint arXiv:1907.11932*, 2019.

[8] D. H. R. Justin M. Rao. The economics of spam. *Journal of Economic Perspectives, 26 (3): 87-110*, 2012.

[9] Z. Kolter and A. Madry. Adversarial robustness: Theory and practice. *Tutorial at NeurIPS*, page 3, 2018.

[10] B. Kuchipudi, R. T. Nannapaneni, and Q. Liao. Adversarial machine learning for spam filters. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–6, 2020.

[11] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984*, 2020.

[12] Q. Liu, R. Zheng, B. Rong, J. Liu, Z. Liu, Z. Cheng, L. Qiao, T. Gui, Q. Zhang, and X.-J. Huang. Flooding-x: Improving bert's resistance to adversarial attacks via loss-restricted fine-tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5634–5644, 2022.

[13] J. X. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *arXiv preprint arXiv:2005.05909*, 2020.

[14] A. Petrosyan. Spam: share of global email traffic 2011-2023. Accessed: 2024-02-15.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (volume 1: long papers)*, pages 856–865, 2018.

[16] R. Stone. Assassin dataset. `https://huggingface.co/datasets/talby/spamassassin`, 2023. Accessed: 2024-03-12.

[17] J. Wei and K. Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[18] Wikipedia contributors. Danskin's theorem – wikipedia, the free encyclopedia, 2024. [Online; accessed 12-April-2024].

# 7 Appendices

**Algorithm 1** Compute importance scores for selecting words in spam emails for replacement

```
procedure TOKEN IMPORTANCE RANKING(tokens, model, attack_type, tokenType)
    Input:
    tokens: the tokens in a spam email
    model: the classifier
    attack_type: "white" or "black"
    token_type: "tf-idf" or "embedding"
    if attack_type == "white" then
        deltas ← attack(tokens, model)                    ▷ Calculate a perturbation delta of each token
        if token_type == "tf-idf" then
            return sort(tokens, by=deltas, order="ascending")                ▷ More negative, more important
        else if token_type == "embedding" then
            return sort(tokens, by=attention_mask * deltas, order="descending")  ▷ Ignore pads, prioritize importance
        end if
    else if attack_type == "black" then
        masked_sentences ← mask(tokens)     ▷ Each token masked individually to create a new sentence
        logits ← model(masked_sentences)
        return sort(tokens, by=logits, order="descending")     ▷ Higher logits indicate higher importance
    end if
end procedure
```

Table 8: The performance of adversarial defense models against other attacks with $k = 0.1$ and $\lambda = 0.1$ for gradient-based white-box defenses

| **Defense Model** | **Attack** | **Accuracy** |
|---|---|---|
| ALBERT + PGD-$L_\infty$ | ALBERT + PGD-$L_2$ | 0.98 |
| | ALBERT + Targeted PGD-$L_\infty$-1 | 0.99 |
| | ALBERT + Targeted PGD-$L_\infty$-2 | 0.98 |
| ALBERT + PGD-$L_2$ | ALBERT + PGD-$L_\infty$ | 0.98 |
| | ALBERT + Targeted PGD-$L_\infty$-1 | 0.99 |
| | ALBERT + Targeted PGD-$L_\infty$-2 | 0.98 |
| ALBERT + Targeted PGD-$L_\infty$-1 | ALBERT + PGD-$L_\infty$ | 1.00 |
| | ALBERT + PGD-$L_2$ | 1.00 |
| | ALBERT + Targeted PGD-$L_\infty$-2 | 1.00 |
| ALBERT + Targeted PGD-$L_\infty$-2 | ALBERT + PGD-$L_\infty$ | 0.98 |
| | ALBERT + PGD-$L_2$ | 0.98 |
| | ALBERT + Targeted PGD-$L_\infty$-1 | 0.99 |
| BERT + Flooding | ALBERT + GPT-4 | 0.96 |
| | LSTM + Text Perturbation | 1.00 |

Table 9: Adversarial examples

| Attack | Original Email | Adversarial Email |
|---|---|---|
| ALBERT + GPT-4 API | You have a new **private message waiting** thank you very much for your **mortgage** refinance application. **The company that best suited** your needs is requesting applicant verification in order to process your claim... | You have a new **confidential communication on hold** thank you very much for your **housing loan** refinance application. **The organization premier fitting** your needs is requesting applicant verification in order to process your **application** claim... |
| MLP + GPT-4 API | **Returned mail**: see **transcript** for **details** the original **message** was received at **Tue**, 19 Jul 2005 13:01:45 +0200 from root@localhost. The following **addresses** had **permanent fatal** errors... | **Undelivered communication**: see **documentation** for **particulars** the original **note** was received at **Tuesday**, 19 July 2005 13:01:45 +0200 from root@localhost. The following **endpoints** had **enduring critical** errors... |
| BERT + BERT | order hydrocodone **online** without prescription - **cheap** hydrocodone . . . need **vicodin ?** phentermine we ' re **fast** . **low price** prescription pain meds . free md consult fast shipping ! no more pain - **get** more info now | order hydrocodone**.** without prescription - **inexpensive** hydrocodone... needed **vicradin =** phentermine we're **shopping**. **large available** prescription pain meds. free md consult fast shipping! no more. - **got** more info now |

Table 10: Baseline model (LSTM) attack configurations

| Config ID | Sentence Shuffling | Synonyms Replacement | Homoglyphs Replacement | Keyboard Typo | Special Character Insertion |
|---|---|---|---|---|---|
| 1 | False | True | False | False | False |
| 2 | True | True | False | False | False |
| 3 | False | False | True | False | False |
| 4 | True | False | True | False | False |
| 5 | False | False | False | True | False |
| 6 | True | False | False | True | False |
| 7 | False | False | False | False | True |
| 8 | True | False | False | False | True |
| 9 | False | True | True | True | True |
| 10 | True | True | True | True | True |

Table 11: Baseline model (LSTM) original performance

| Model | Dataset | Original Accuracy |
|---|---|---|
| Original LSTM | Test | 98.37% |
| Original LSTM | Spam | 97.45% |

Table 12: Defending Baseline Model (LSTM) Performance Against Test/Spam Dataset

| Model | Dataset | Robust Accuracy |
|---|---|---|
| Defending LSTM | Test | 98.28% |
| Defending LSTM | Spam | 97.61% |

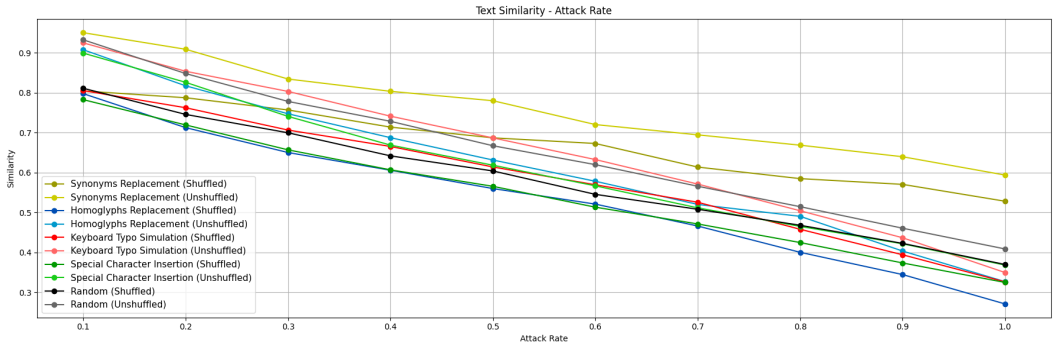


Figure 2: Attacked baseline model (LSTM) text similarity vs attack rate

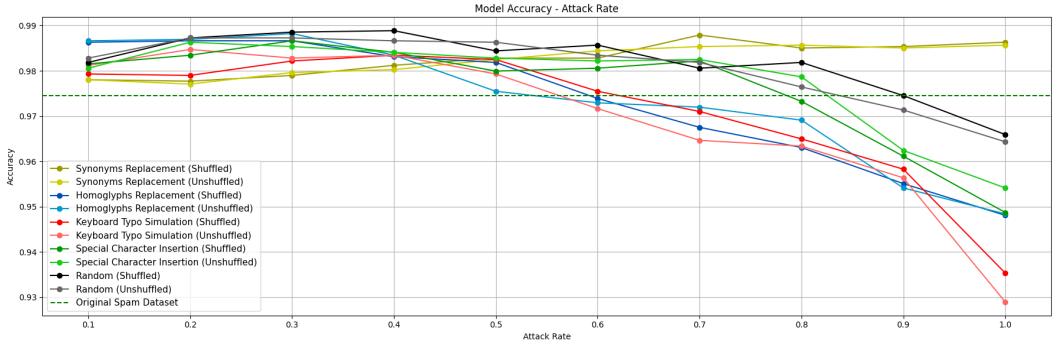


Figure 3: Defending baseline model (LSTM) classification accuracy vs attack rate

Table 13: Baseline model (LSTM) adversarial training comparison (attack rate 0.8)

| **Attack Config** | **Without Adversarial Training** | **With Adversarial Training** |
|---|---|---|
| Synonyms Replacement (Shuffled) | 93.43% | 98.50% |
| Synonyms Replacement (Unshuffled) | 96.02% | 98.57% |
| Homoglyphs Replacement (Shuffled) | 80.99% | 96.31% |
| Homoglyphs Replacement (Unshuffled) | 92.39% | 96.91% |
| Keyboard Typo Simulation (Shuffled) | 55.25% | 96.50% |
| Keyboard Typo Simulation (Unshuffled) | 65.10% | 96.34% |
| Special Character Insertion (Shuffled) | 52.26% | 97.32% |
| Special Character Insertion (Unshuffled) | 64.65% | 97.87% |
| Random (Shuffled) | 59.62% | 98.18% |
| Random (Unshuffled) | 71.56% | 97.64% |