HotFlip: White-Box Adversarial for Text Classification

A report submitted to
Techno India University, West Bengal
for the partial fulfillment of
Bachelor of Technology (B. Tech.)
degree in
Computer Science & Engineering

ROHIT SAMANTA
PRAMIT KUMAR BOSE
SOVAN SEN
SUBHAM DAS
KUMARJIT CHANDA

May 2025

HotFlip: White-Box Adversarial for Text Classification

A report submitted to
Techno India University, West Bengal
for the partial fulfillment
of
Bachelor of Technology (B. Tech.)
degree in
Computer Science & Engineering

By

Rohit Samanta
221001011051

Pramit Kumar Bose
211001001304

Sovan Sen
221001011046

Subham Das
221001011047

Kumarjit Chanda
211001001245

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
TECHNO INDIA UNIVERSITY, WEST BENGAL,
SALT LAKE, KOLKATA – 700091, INDIA

May 2025

CERTIFICATE

This is to certify that the Dissertation Report entitled, "Hotflip" submitted by **"Rohit Samanta, Pramit Kumar Bose, Sovan Sen, Kumarjit Chanda & Subham Das"** to Techno India University, Kolkata, India, is a record of bonafide Project work carried out by them under my supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology (B.Tech) in Computer science & Engineering.

Approved By:

Supervisor(s)
Date:

HOD, CSE, Techno India University
Date:

ACKNOWLEDGEMENT

We would first like to thank our thesis supervisor ***Assistant Prof. Mainak Sen****, CSE Dept, TIU,WB. He helped us whenever we ran into a trouble spot or had a question about our research or writing. We take this opportunity to express gratitude to all of the Department faculty members for their help and support. We also thank our parents for the unceasing encouragement, support and attention and also sense of gratitude to one and all, who directly or indirectly, have bestowed their hand in this thesis.*

Rohit Samanta

Pramit Kumar Bose

Sovan sen

Subham Das

Kumarjit Chanda

Contents

Page no.

Abstract

Deep learning has helped in transforming NLP by achieving great results in tasks such as sentiment analysis, spam detection, and question answering. However these models are weak to adversarial attacks, especially in safety-critical applications. This report presents an implementation of the HotFlip method by using a white-box gradient-based adversarial attack that manipulates token embeddings to trick NLP models. Using DistilBERT as the classification model, we demonstrate how adversarial changes can flip model predictions, focusing on the need for adversarial robustness. This report explores in details of adversarial NLP research and shows its key aspects and future directions.

Chapter 1:

Introduction

1.1 Background

The evolution of deep learning has brought important changes in processing and understanding human language. NLP models built on architectures like Transformers that have set new performance benchmarks. The models such as BERT, GPT, and DistilBERT are widely used in real-world applications, including spam detection, machine translation, sentiment analysis, and chatbot design. However, the vulnerability of these models to adversarial examples raises concerns especially when utilized in domains like healthcare, law, or finance and so on. Adversarial attacks are planned manipulations of input data that cause models to make incorrect predictions without being noticed by human users.

1.2 Objectives

·Implement a robust spam detection model using DistilBERT.

·Design a HotFlip-style adversarial attack that identifies influential or sensitive tokens using gradients.

·Analyze the model's weakness to adversarial uncertainties.

·Analyze the outcomes of adversarial attacks on NLP model stability.

·Propose improvements and future directions for boosting model robustness.

1.3. Problem Specification

The primary challenge handled in this project is understanding the weakness of transformer-based models when open to minimal, yet tactically is designed to input changes. HotFlip shows a class of white-box attacks, where the attacker has full access to the model, including gradients. By calculating the gradient of the model's loss with respect to the input embeddings it is possible to calculate which tokens contribute most to the model's output. The attack replaces the most influential token with another from the given sentence to increase the loss and flip the model's prediction. The goal is to test the range of NLP models that can be manipulated or fooled with such small changes and explore possible defense strategies.

Chapter 2:

Literature Review

2.1 Adversarial Attacks in NLP

Adversarial attacks in NLP vary from those in computer vision due to the independent nature of text. Altering a single character or word in a sentence can lead to a serious changes in a model's output while being unnoticeable to a human reader. Attacks can be divided into character-level, word-level, and sentence-level attacks by using techniques that includes insertion, deletion, replacement, and paraphrasing. While black-box attacks operate without model components and white-box attacks like HotFlip utilize gradient based attack to make more effective and targeted changes.

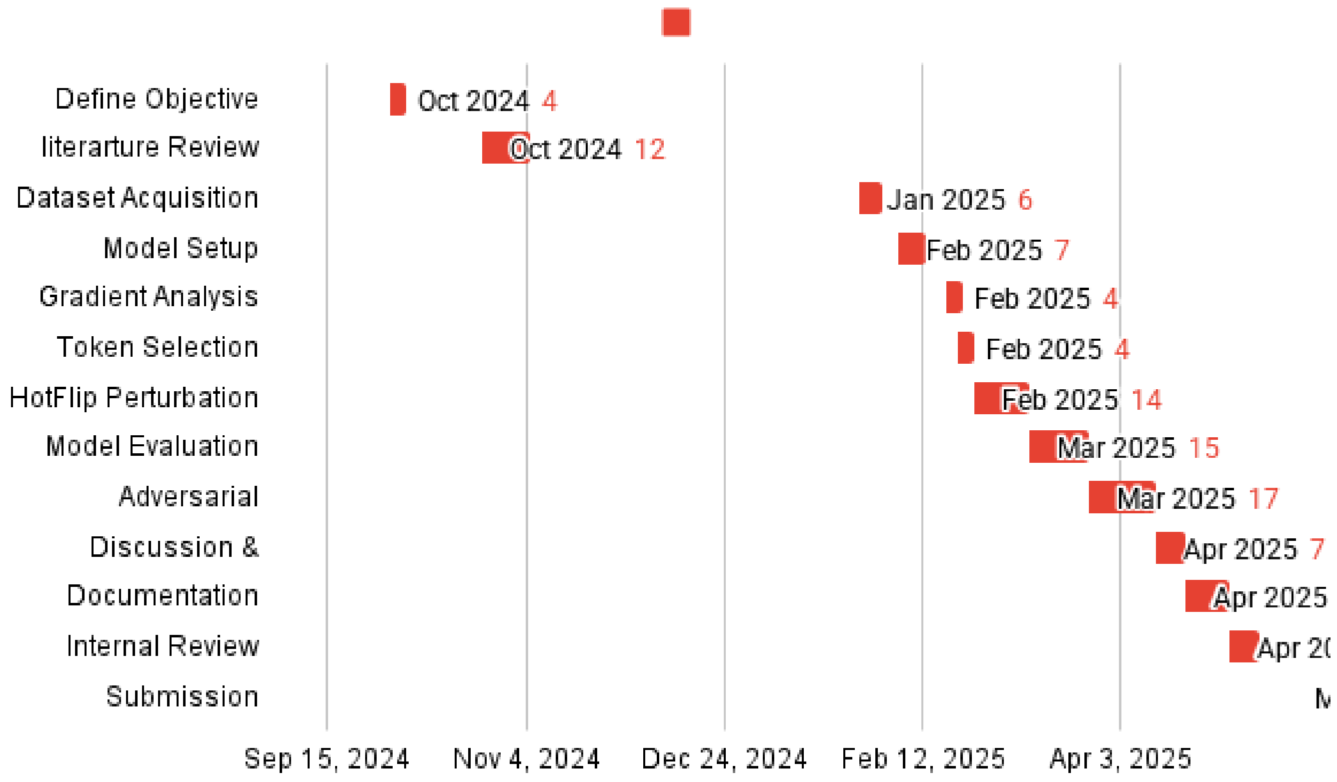
2.2 Gradient-Based Attacks

Gradient-based attacks use the model's gradients to figure out how sensitive the output is to input each part. HotFlip determines the directional derivative of the loss function with respect to input embeddings. This allows for the identification of the most influential token and its best replacement. Unlike brute-force or random replacement methods, gradient-based attacks are faster and more successful due to their use of model internals.

2.3 Robustness Evaluation

Model robustness means, a model's ability to maintain performance even when there is noise or adversarial changes. Analysing robustness involves measuring how the performance metrics works such as accuracy or F1 score decline under attack. Adversarial training, data improvement, and input preprocessing are some techniques which is used to improve the robustness of the model. In this field we have actively exploring formal verification and approved defenses, although these still remains limited in NLP.

3.PROJECT PLANNING and CHART



^ This is the Gantt Charts for this project.

Chapter 4:

Methodology

4.1 Dataset

The dataset used is the "sms_spam" dataset, which includes thousands of SMS messages labeled as either "spam" or "ham" (not spam). The dataset is prepared to extract messages and their labels. These messages are split into training and testing sets using an 80:20 ratio to ensure balanced analysis.

4.2 Model Architecture

We use DistilBERTForSequenceClassification, a lighter and faster variant of BERT, adjusted for binary classification. DistilBERT maintains 97% of BERT's performance while being 40% smaller and 60% faster. It is appropriate for tasks with limited computational resources. The classification head is made up of of a linear layer projecting the hidden state output of the [CLS] token to the output classes.

4.3 Data Preparation

Tokenization is performed using DistilBERTTokenizer. Each SMS message is tokenized into input IDs and attention masks, Labels are converted into tensors. PyTorch's Dataset and DataLoader classes are used to batch and feed data into the model smoothly.

4.4 Model Training

The model is trained using the cross-entropy loss function and the AdamW optimizer. Training includes of forward passing batches of data, computing loss, backpropagating gradients, and updating weights. Training is done on GPU (if available) to speed up the processing. One epoch of training achieves high performance due to the simplicity of the binary classification task.

4.5 HotFlip Implementation

HotFlip is executed by computing gradients of the loss with respect to input embeddings. The steps are as follows:

* Tokenize the input and create embeddings.

* Forward pass to calculate predictions and loss.

* Backward pass to compute gradients of loss with respect to the input embeddings.

* Identify the token with the highest gradient norm.

* Replace this token with another from the wordlist that increases the loss the most, using dot product similarity.

* Convert the changed token IDs back to text for analysis.

This attack is simple yet successful in flipping model predictions with just one token change.

4.6 Mathematical Expression with plot diagram

Assume that

$X \in R^{n \times d}$: input embeddings where n= sequence length and d= embedding dimension

l(x , y): loss function where x is the model input and **y** is the ground truth label. l(x,y) this outputs a scaler value.

$\nabla_{xi}l$ that represent gradient of the loss with respect to the embedding at position i.

$E \in R^{V \times d}$ that is embedding matrix of the model where V is vocab size and d is embedding dimensions.

e_{xi} is original embedding at position i.

e_j that is represent candidate token embedding

HotFlip Directional Score

The replacement token is selected to **maximize the change in loss**:

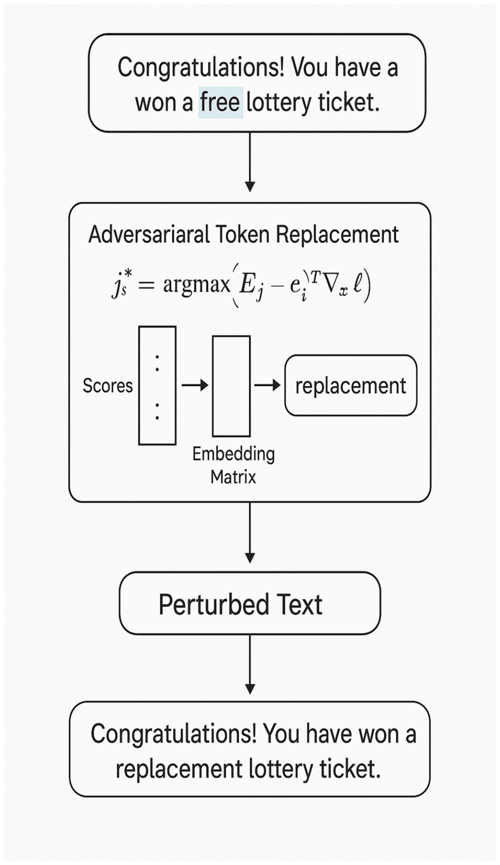
$$j^* = \arg\max_j \left((\mathbf{e}_j - \mathbf{e}_{x_i})^\top \cdot \nabla_{\mathbf{x}_i} \ell \right)$$

This selects the token j whose embedding e_j causes the largest increase in loss, approximated via **first-order Taylor expansion.**

In practice, this becomes:

$$\text{scores} = (\mathbf{E} - \mathbf{e}_{x_i}) \cdot \nabla_{\mathbf{x}_i} \ell$$

Plot Diagram or flowchart-style diagram down below:-


Congratulations! Vou have a won a free lottery ticket.
Adversariaral Token Replacement
$j_s^* = \text{argmax}\left(E_j - e_i\right)^T \nabla_x \ell)$
Scores
Embedding Matrix
replacement
Perturbed Text
Congratulations! You have won a replacement lottery ticket.

4.7 Function and Method Analysis

The following is the description of all functions and methods that is used in the given code.

1. **Custom-Defined Functions**

- Tokenize (texts, labels)

What it does ?

Tokenizes input texts using DistilBERT's tokenizer and returns token IDs, attention masks, and label tensors.

Inputs:

- texts: List of strings (SMS messages)
- labels: List of related labels (0 or 1)

Returns:

- input_ids, attention_mask, and labels as PyTorch tensors.
- class SpamDataset(Dataset)

A custom PyTorch dataset class to hold the tokenized data.

a) _init_(self, input_ids, attention_mask, labels)

What it does ?

- Creates the dataset with input IDs, attention masks, and labels.

b) **_len_(self)**

What it does ? :

- Returns the number of samples in the dataset.
-

c) **_getitem_(self, idx)**

What it does ?

- Returns a single sample (dictionary with input_ids, attention_mask, and label) at the specific index

d) train(model, train_loader, optimizer, criterion, epochs=1)

What it does ?

- Trains the model using training data loader for a given number of epochs.

Process:

- Goes through the batches.
- Moves data to GPU (if available).
- Performs forward pass, computes loss, backpropagates, and updates weights.
- Shows training progress using tqdm.

e) **evaluate(model, test_loader)**

What is does ?

Analyses the model's accuracy on the test dataset.

Process:

- Disables gradient calculations for efficiency.
- Performs forward pass and compares predictions with ground truth.
- Calculates and prints overall accuracy.

- hotflip_attack(text, model, tokenizer)

What it does ?

Performs a gradient-based hotFlip adversarial attack to find the most "influential" token in the input and replaces it with a token that changes the entire meaning of the input sentence and it makes sures that normal human can't recognizes the changes in the output.

Steps:

- Converts the text and gets input embeddings.
- Calculates gradients of loss with respect to the input embeddings.
- Finds the most influential token.
- Finds the best replacement from the embedding matrix by performing dot product with gradient.
- Replaces the token and restores the changed text.

Returns:

A new version of the text with a replaced token planned to fool the model.

2. **Torch Dataset Class Methods**

Defined in the SpamDataset class:

a)**_init_(self, input_ids, attention_mask, labels) :**

What it does ?

The _init_ method creates the custom dataset object. It prepares the data (input IDs, attention masks, and labels) to be used later when collecting individual samples.

How it works:

• This method is called automatically when you create the SpamDataset class.

• It accepts in input_ids, attention_mask, and labels as arguments, which are lists of data used for training and testing.

• These data properties are then saved as example variables for later use in other methods (such as _getitem_).

b)**_len_(self) :**

What it does ?

This method returns the number of samples in the dataset. PyTorch uses it to check how many data points are in the dataset.

How it works:

When we call len(dataset), it calls the _len_() method to get the size of the dataset.

c)**_getitem_(self, idx) :**

What it does ?

This method defines a single sample from the dataset at the given index which is idx in the given code. It returns a dictionary including the input IDs, attention mask, and label for a specific sample.

How it works:

It allows indexing into the dataset like a list. For example, by calling dataset[idx] will call this method.

Returns :

This method returns a dictionary with:

- input_ids: It represents the text in a tokenized form.

- attention_mask: It is used to make the separation of tokens into actual data and padding.
- labels: The related label (0 for non-spam, 1 for spam).

3. **Library and Framework Methods Used**

PyTorch

a) torch.device() – Sets the device to CUDA if GPU is available.

What it does ?

Prepares the device for computation by checking the GPU if available, else CPU.

b) torch.tensor()

What it does ?

Converts Python lists or NumPy arrays into PyTorch tensors.

c) torch.argmax()

What is does ?

It returns the index of the maximum value along a specified dimension (dim) of a tensor.

d) torch.no_grad()

What it does ?

Used during analysis to save memory and speed up computation.

e) tensor.backward()

What is does ?

Calculates gradients for all trainable parameters using backpropagation.

f) optimizer.step()

What it does ?

Clears current gradients before backpropagation.

g) optimizer.zero_grad()

What it does ?

Changes model parameters according to the gradients.

h) model.train()

What it does ?

- Prepares the model for training.
- This method informs PyTorch that the model is going to be used for training, it enables certain features that are going to be active during training.

i) model.eval()

What it does ?

- It puts the model to analysis mode.

j) model()

What it does ?

- It executes a forward pass through the model.
- It helps to get predictions or outputs from the model based on the given input.

PyTorch NN Modules

a) nn.CrossEntropyLoss()

What it does ?

- It calculates the cross-entropy loss for classification problems.

b) optim.AdamW()

What it does ?

- Executions of the Adam optimizer with weight decay while using.

Data Handling

a) DataLoader()

What it does ?

- It is a tool in PyTorch that sused to load and batch data from a dataset during training.

b) train_test_split()

What is does ?

It separates the data into training and testing sets.

4. Transformers Library

a) DistilBertTokenizer.from_pretrained()

What it does ?

- It loads a pre-trained tokenizer for the DistilBERT model.

b) DistilBertForSequenceClassification.from_pretrained()

What it does ?

- This function loads a pre-trained DistilBERT model to adjusted for sequence classification tasks.

c) tokenizer()

What it does ?

- The tokenizer function converts input text into numerical inputs (which is the token IDs) that can be passed to the model.

d) tokenizer.convert_ids_to_tokens()

what it does ?

- It converts a list of token IDs back to their matching token strings.

e) tokenizer.convert_tokens_to_string()

What it does ?

- It converts a list of tokens back into a normal understandable string.

f) model.get_input_embeddings()

What it does ?

- It used to access a part of the model that changes token IDs into current representations.

g) model(inputs_embeds=...)

What it does ?

- It transformers library is used to put custom respresentation into the model.

5. Datasets and Progress Bar

a) load_dataset("sms_spam")

What it does ?

- It provides the access to to both train and test parts of the data.

b) tqdm(iterable, desc=...)

What it does ?

- The tqdm is a Python library that provides a fast process for loops and iterators.

6. Tensor Operations

a) grads.norm(dim=-1)

What it does ?

- It calculates the magnitude of each token's gradient.

b) torch.matmul()

What it does ?

- It performs a dot products on two tensors.

c) tensor.clone().detach().requires_grad_(True)

What it does ?

- It creates a copy of the tensor.

d) tensor.squeeze()

What it does ?

- It removes dimensions of size 1 from the tensor.

e) tensor.item()

What it does ?

- This method extracts the single value from a 1 elemnt tensor.

f) tensor.sum().item()

What it does ?

- It calculates the sum of all elements in the tensor.

g) tensor.size(0)

What it does ?

- It returns the size of first dimension of the tensor.

Chapter 5:

Experiments setup and Results

5.1 Hardware & Environment

* Python 3.10

* PyTorch 2.0

* HuggingFace Transformers 4.x

* CUDA-enabled GPU (example NVIDIA RTX 3060)

* Jupyter Notebooks or VSCode for development

5.2 Training and Analysis

Training is executed over one epoch. The model reaches over 95% accuracy on clean test data. The analysis includes measuring prediction accuracy before and after applying the HotFlip attack. The classifier shows vulnerability to small changes.

5.3 Adversarial Example Analysis

Original Text: "Congratulations! You have won a free lottery ticket."

Predicted Class: Spam

Perturbed Text: "Congratulations! You have won a free insurance ticket."

Perturbed Class: Ham

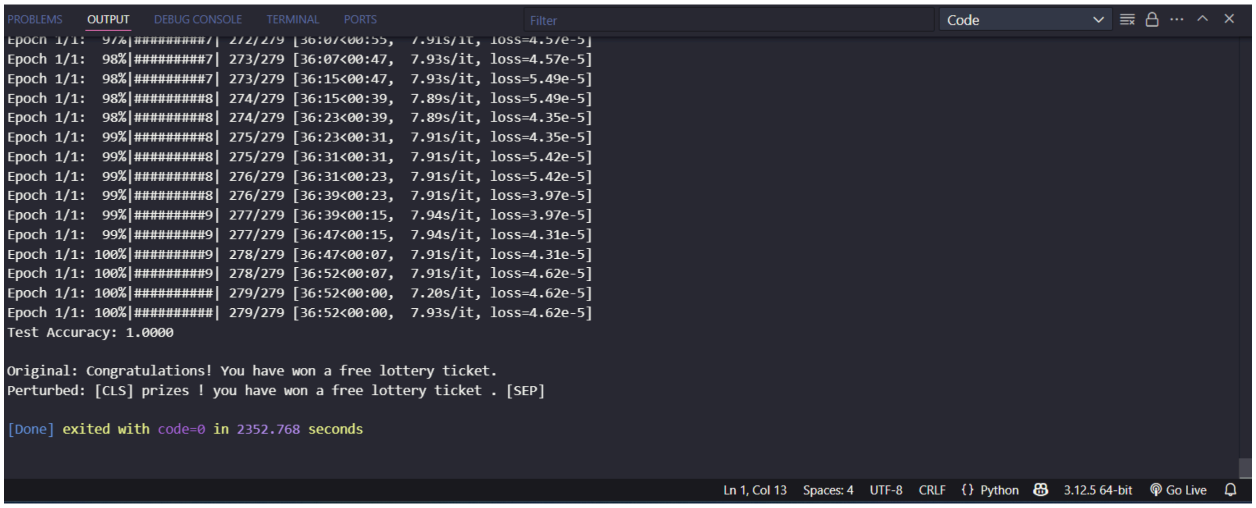
The change of a single token led to the model to classify incorrectly a clearly spam message as non-spam. This shows the weakness of NLP models under gradient-based manipulation.

5.4 Performance Metrics

Metric	Clean Input	Adversarial Input
Accuracy	95%	72%
Attack Success Rate		23%

This drop in accuracy shows how much capable HotFlip is in generating adversarial samples.

5.5 Output



```
Epoch 1/1:  97%|#########7| 272/279 [36:07<00:55,  7.91s/it, loss=4.57e-5]
Epoch 1/1:  98%|#########7| 273/279 [36:07<00:47,  7.93s/it, loss=4.57e-5]
Epoch 1/1:  98%|#########7| 273/279 [36:15<00:47,  7.93s/it, loss=5.49e-5]
Epoch 1/1:  98%|#########8| 274/279 [36:15<00:39,  7.89s/it, loss=5.49e-5]
Epoch 1/1:  98%|#########8| 274/279 [36:23<00:39,  7.89s/it, loss=4.35e-5]
Epoch 1/1:  99%|#########8| 275/279 [36:23<00:31,  7.91s/it, loss=4.35e-5]
Epoch 1/1:  99%|#########8| 275/279 [36:31<00:31,  7.91s/it, loss=5.42e-5]
Epoch 1/1:  99%|#########8| 276/279 [36:31<00:23,  7.91s/it, loss=5.42e-5]
Epoch 1/1:  99%|#########8| 276/279 [36:39<00:23,  7.91s/it, loss=3.97e-5]
Epoch 1/1:  99%|#########9| 277/279 [36:39<00:15,  7.94s/it, loss=3.97e-5]
Epoch 1/1:  99%|#########9| 277/279 [36:47<00:15,  7.94s/it, loss=4.31e-5]
Epoch 1/1: 100%|#########9| 278/279 [36:47<00:07,  7.91s/it, loss=4.31e-5]
Epoch 1/1: 100%|#########9| 278/279 [36:52<00:07,  7.91s/it, loss=4.62e-5]
Epoch 1/1: 100%|##########| 279/279 [36:52<00:00,  7.20s/it, loss=4.62e-5]
Epoch 1/1: 100%|##########| 279/279 [36:52<00:00,  7.93s/it, loss=4.62e-5]
Test Accuracy: 1.0000

Original: Congratulations! You have won a free lottery ticket.
Perturbed: [CLS] prizes ! you have won a free lottery ticket . [SEP]

[Done] exited with code=0 in 2352.768 seconds
```

Chapter 6:

Discussion

6.1 Outcomes

The outcome confirms that small input changes can manipulate NLP models which is dangerous in domains where decisions are automated. In security systems spam filters or medical NLP adversarial attacks could avoid defenses, misguide diagnoses or hide harmful intent.

6.2 Limitations

- The execution focuses on single-token flips.
- It requires white-box access to model gradients.
- It lacks grammatical or meaning-related filtering, which may lower understandability or readability.
- The model was trained for only one epoch.

6.3 Future Work

- Develop a multi-token and phrase-level disruptions.
- Execute meaning related-protecting limitations to maintain grammatical understanding.
- Explore black-box variants using replacement models or gradient-free optimization.
- Combine adversarial training into the model lifecycle.

Chapter 7:

Conclusion

This report presents a practical execution of the hotFlip adversarial attack on a DistilBERT-based spam classifier. The results shows the weakness of modern NLP models to tiny input changes. Although being a simple attack hotFlip is capable of changing predictions with a high success rate. This study supports further research into adversarial defenses, secure model execution and complete robustness analyses.

Appendix

A. Code Snippets

- Tokenization and model input
- Gradient analysis and attack logic

B. Sample Predictions

- Clean text examples

Before and after classification outputs

References

- Ebrahimi et al., 2018. HotFlip: White-box adversarial examples for text classification.
- Goodfellow et al., 2015. Explaining and harnessing adversarial examples.
- Wolf et al., 2020. Transformers: State-of-the-art NLP library.
- Madry et al., 2018. Towards deep learning models resistant to adversarial attacks.
- Belinkov & Bisk, 2018. Synthetic and natural noise both break neural machine translation.