A DistilBERT-Based NLP Model for Detecting Malicious URLs

1st Ritaj Muhanna Alhamli
Dept. Computer Engineering
Imam Abdulrahma Bin Faisal University
Dammam
2210002809@iau.edu.sa

2nd Nora Abdullah Aljomuh
Dept. Computer Engineering
Imam Abdulrahma Bin Faisal University
Dammam
2220004452@iau.edu.sa

3rd Mozoon Alkhalis
Dept. Computer Engineering
Imam Abdulrahma Bin Faisal University
Dammam
2220002873@iau.edu.sa

***Abstract*—Malicious URLs are one of the most common ways attackers launch phishing campaigns, spread malware, or host defaced websites. Because these URLs are constantly changing and often designed to look harmless, traditional detection methods struggle to keep up. In this project, we built an NLP-based classifier using DistilBERT to automatically identify malicious URLs. We used the "Malicious URLs Dataset" from Kaggle, which contains over 650K URLs labeled as benign, phishing, malware, or defacement. Minimal preprocessing was required since the URL structure itself holds important patterns, and DistilBERT's tokenizer handled the necessary text processing. The model was fine-tuned for three epochs using AdamW with a 2e-5 learning rate and a batch size of 16. Our evaluation on a held-out test set showed strong performance, achieving an accuracy of 98.89% and consistently high precision, recall, and F1-scores across all four classes. We also implemented a simple prediction function that allows users to classify any URL interactively. Overall, the results confirm that a lightweight transformer model like DistilBERT can effectively learn URL patterns and provide reliable, real-time threat detection.**

Keywords—malicious URLs, phishing detection, NLP, DistilBERT, cybersecurity, URL classification

I. INTRODUCTION

The internet has become central to almost everything we do, but it also brings a growing list of security risks. One of the simplest and most common ways attackers reach victims is through malicious URLs. These links may lead to phishing pages, distribute malware, or redirect users to defaced websites. Because attackers constantly change and disguise their URLs, relying on manual inspection or basic blacklist systems is no longer enough.

With recent advancements in Natural Language Processing (NLP), especially transformer models, it is now possible to analyze URLs the same way we analyze text. Instead of manually creating rules or extracting handcrafted features, these models can learn patterns directly from the data. This makes them a strong option for building an automated URL threat-detection system.

II. OBJECTIVES & GOLES

The main objectives of this project are:

- ✓ To build an NLP model using DistilBERT for URL classification.
- ✓ To train and evaluate the model on a labeled malicious-URL dataset.
- ✓ To measure how well the system performs using standard metrics.
- ✓ To explore whether transformer models can effectively replace traditional feature-engineering methods for malicious URL detection.

III. RELATED WORK

Natural Language Processing (NLP) is one of the main game changers in combating phishing and malicious cyber activities. By analyzing and decoding URL patterns, NLP can capture the lexical structure that's most used with phishing. Traditional machine learning models such as Support Vector Machines (SVM), Logistic Regression, and Random Forests have been applied using handcrafted URL features. Furthermore, these classical approaches struggle with obfuscation tactics used by attackers, including homoglyphs and disguised subdomains, which require more manual feature engineering. This limitation has driven the shift to automated deep learning approaches that can learn directly from URLs.

A. Tokenization and Representation of URLs

Tokenization plays an important role in applying NLP techniques to cybersecurity tasks such as phishing URL detection. Unlike natural language text, URLs exhibit a specific structure and patterns consisting of multiple segments, including the protocol, domain name, path, and parameters. Frequently, these segments contain abbreviations, concatenated terms, random character sequences, and symbolic delimiters. Consequently, phishing URLs can hide malicious intent through lexical obfuscation, homoglyphs, and brand impersonation embedded within subdomains or directory paths. Traditional feature-based approaches are typically based on shallow lexical patterns, such as URL length or keyword presence, which are easily manipulated by attackers. Recent studies specifically discuss tokenization challenges for URL semantics using transformer-based models. The work by [1] introduces URLTran, which examines various tokenization strategies designed to maximize contextual learning from URL structures. Illustrating excellent classification performance across low false-positive thresholds. Similarly, El Mahdaouy

et al. propose DomURLs-BERT, combining URL-aware preprocessing with dedicated domain and path markers such as [DOMAIN], [PATH] to preserve structural intent and improve representational clarity during masked language modeling [2]. Overall, these methods show that maintaining linguistic structure is vital for transformers to capture contextual dependencies in phishing URLs.

B. NLP-Driven URL Classification Using Transformer Models

Nowadays, recent studies have established transformer architectures as highly effective in phishing URL due to their perfect performance in extracting contextual meaning from sequential text. URLTran is originally one of the first works to apply pretrained language models such as BERT and RoBERTa directly to URLs, proving true positive rates and showing robustness to adversarial perturbations such as homoglyph substitutions and token splitting [1]. The following figure demonstrates the design of the URLTran phishing URL detection model, which we can conclude that phishing detection benefits from deep contextual NLP modeling rather than reliance on surface-level lexical cues.

Furthermore, SecureNet further expands the use of transformers to multiple cyber-text channels, confirming that models such as DeBERTa achieve strong accuracy and generalization when identifying wicked behavior in diverse phishing content [3]. In addition, DomURLs-BERT improves URL understanding by incorporating URL-specific preprocessing and tokenization, achieving superior performance across multiple malicious URL classification tasks [2]. All these previous studies show that transformer-based NLP dramatically advances phishing detection by enabling more in-depth interpretation of URLs and evolving attack patterns.

C. Hybrid NLP Approaches with Embedded Text

While many phishing detection techniques focus only on the main URL of a webpage, new attacks increasingly camouflage corrupted links within legitimate content elements such as hyperlinks, JavaScript code, and iframe sources. These old-fashioned techniques make traditional URL-only analysis not enough, as a webpage might initially appear benign while still containing harmful embedded redirection paths. To address this challenge, recent NLP-based approaches extend URL classification to include all textual URLs present within a webpage's structure.

PhishTransformer is an example of a hybrid approach. Instead of analyzing only the primary URL, the model gathers every URL reference embedded in the webpage content and applies a combined CNN-Transformer to classify the webpage based on the collective linguistic behavior of these elements. This enables better and stronger detection against advanced attack types such as Browser-in-the-Browser and clickjacking, where malicious resources are nested within trusted domains [4]. By implementing contextual NLP across multiple internal URLs, the system can detect linguistic conflicts and deception patterns even when the accessed webpage itself initially appears legitimate.

D. Research Gaps and Motivation

Although NLP-based methods have improved phishing URL detection, existing approaches still face challenges. Traditional models rely on handcrafted lexical features that attackers can easily bypass. Transformer models better capture URL semantics but often require high computational resources, limiting real-time deployment [1]–[3]. Additionally, some techniques focus only on the main URL and miss malicious links hidden within webpage content [4].

Therefore, our contribution is to build a lightweight transformer model that efficiently understands URL linguistic patterns while supporting scalable deployment is needed. This motivates the use of DistilBERT in this research for building a cybersecurity model with a user-friendly interactive tool.

IV. METHODOLOGY

This section explains how we built and trained our malicious URL classification system using DistilBERT. The workflow includes preparing the dataset, processing the URLs, encoding them for the model, setting up the architecture, training it, and finally evaluating its performance. Our goal was to create a model that is accurate, efficient, and straightforward to use, especially for real-time or practical applications.

A. System Overview

Our approach follows a typical NLP pipeline but with a focus on URL data. We start by preparing the dataset, then rely on DistilBERT's tokenizer to handle most of the text processing. After that, we fine-tune DistilBERT as a multi-class classifier and evaluate its performance on unseen data. Since URLs are short and often contain hidden patterns, using a transformer model allowed us to capture these details more effectively than traditional methods.

B. Dataset Preparation

For this project, we chose a public dataset available on Kaggle named "*Malicious URLs dataset,"* which is a multi-class dataset containing 651,191 URLs divided into four classes: benign, phishing, malware, and defacement URLs. The dataset is in one CSV file and not initially split; hence, we loaded the CSV file and did a split: 70/15/15 train, validate, and test set. The dataset is naturally imbalanced, so we needed to do a stratified split to ensure that the classes remain consistent between the train, validation, and test sets. Fig. 1 is computed by NumPy, the library, and printed using `matplotlib.pyplot` library shows the class distribution across the splits.

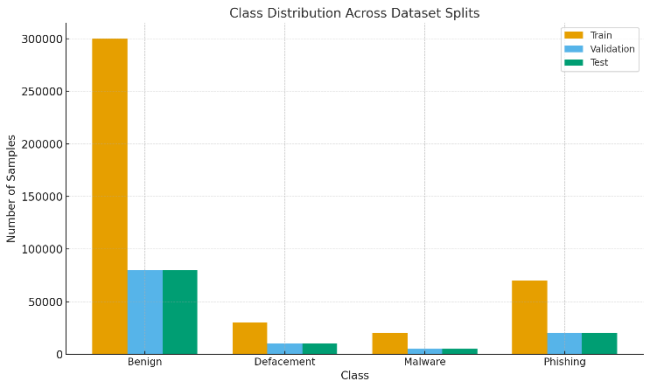

Figure 1. Class distribution across the stratified train, validation, and test splits

C. Pre-processing

The dataset includes two columns: one is the URLs, and the other is the classes. So manual preprocessing wasn't needed. All the URL details are important in classifying it, as altering it would not be a smart move. DistilBERT's tokenizer does the main methods automatically, that is, segmenting the raw URL into parts that the model can grasp, converting these parts into numbers, and padding these parts to have the same length. Our chosen method of pre-processing guarantees that our URLs remain in the required structure. More details about tokenization will be explained in the next subsection.

D. Tokenization and Input Encoding

Tokenization is the process of making the URL input fit the DistilBERT model. It begins with splitting the URL into meaningful parts, for example, the domain name, the extension, and most importantly, suspicious patterns if they exist. Then, each part is mapped to a numerical value from the DistilBERT's vocabulary. And since DistilBERT only accepts a fixed-length input, the tokenizer pads the shorter parts and truncate the longer ones, making all parts fit into a unified length which is 64 tokens. Finally, the tokenizer also utilizes an attention mask, which identifies which parts are the padded sections to ignore and which contain the actual pattern to focus on. These encoded inputs (input_ids and attention_mask) are the final input that the model actually sees and uses during training, prediction, and evaluation.

E. Model Architecture

For the model, we chose DistilBERT for a number of reasons. First, it provides us with the power of a transformer model while still being fast and lightweight compared to the traditional BERT model. Additionally, it captures patterns well, even in small text sequences such as URLs. Hence, it balances between accuracy and efficiency, making it the best applicable choice for our project. Fig. 2 mainly illustrates a comparison between BERT and the DistilBERT architectures. The primary difference is that DistilBERT has six transformer layers instead of twelve, which translates into it being a lighter and faster model.

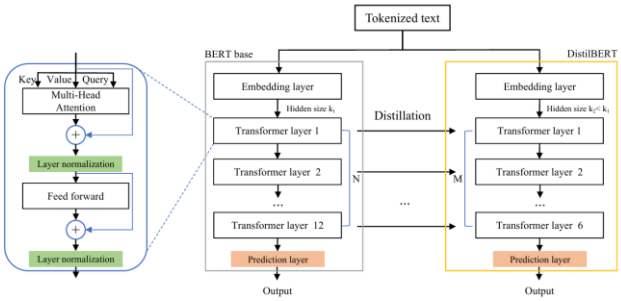

Figure 2. The DistilBERT model architecture and components (Reproduced from [5]).

In addition, DistilBERT is created by a distillation process, where a smaller model "the student" learns to mimic and copy the larger model "the teacher". Thus, the student learns the important internal behaviour while ignoring and removing complex, unnecessary elements. This is why DistilBERT excels in a task such as classifying URLs; it is fast and yet still efficient.

After discussing the chosen model architecture and giving reasons answering why we chose it. A code snippet from our project notebook exhibits the process of loading the pre-trained DistilBERT model and preparing it for our URL classification task. We used the "distilbert-base-uncased" version, and since our dataset has four classes, we set the number of labels to four. The model is now ready and prepared to be fine-tuned on our URL data.

```
# Load DistilBERT
from transformers import
DistilBertForSequenceClassification

model =
DistilBertForSequenceClassification.from_pret
rained(
    "distilbert-base-uncased",
    num_labels=4
)
```

F. Training Setup

In this step, we define the training arguments, which include specifying the model's parameters, such as the number of epochs, batch size, the learning rate, and other parameters used for fine-tuning the model. We trained the model for three epochs with the AdamW optimizer, which is commonly used in transformer models. The batch size used is 16; this size maintains memory usage and the model's performance. The learning rate is 2e-5, which is a typical choice for fine-tuning the model. Weight decay is used to prevent overfitting, which works by penalizing large weights. All of these arguments guide the model to train and learn efficiently. These discussed settings are shown in the following code snippet from our notebook.

```
# Training arguments
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="results",
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_steps=100,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    learning_rate=2e-5,
    weight_decay=0.01,
    report_to=[]  # disables wandb
)
```

G. Training Process

In this step, we create the trainer object and apply all the training configurations to actually train the model. We used the *trainer.train(),* and we monitored the progress of the model via the progress bar, the training, and the validation loss. The following snippet of code shows the training process. From the training output, we can see that the training loss started relatively higher than the 2nd epoch and continued to decrease. This clearly demonstrates that the model is successfully learning patterns of the URL. Additionally, the validation loss remained stable and similar to the training loss across all three epochs, which confirms the model's generalization abilities on unseen URLs and suggests that the model is not overfitting. Overall, this printed output proves that the training process executed correctly and that the model was improving with each epoch.

```
trainer.train()
```

Table 1. Code snippet of the Training output

Epoch	Training Loss	Validation Loss
1	0.063500	0.053404
2	0.030000	0.021935
3	0.011700	0.009815

H. Model Evaluation

For model evaluation, we tested the model on the unseen test set, which is 15% of the dataset. We evaluated it using a *trainer.evaluate(test.ds).* This step is essential to exhibit the model's ability to classify the URL into benign, phishing, malware, and defacement classes. The evaluation output showed a low evaluation loss, which confirms that the model maintained its performance even on unseen URLs. Moreover, to get a deeper understanding of the model's competence, we computed accuracy, precision, recall, and F1-score for each of the four classes. Table 2 shows a table of the calculated evaluation metrics.

Table 2. Classification Performance of the DistilBERT Model.

Class	Precision	Recall	F1-Score
Phishing	0.9641	0.9596	0.9619
Benign	0.9922	0.9944	0.9933
Defacement	0.9974	0.9991	0.9982
Malware	0.9908	0.9705	0.9805
Macro Avg	**0.9861**	**0.9809**	**0.9835**

The model achieved an overall accuracy of 98.89%, and the F1-score was high across all classes; hence, there is a balance between precision and recall in all classes. This indicates that the model identifies URLs and classifies them correctly in all classes. Additionally, the macro average F1-score was also high, which confirms that the model is consistent and balanced in all classes equally.

I. Prediction Function

We implemented a prediction function to allow the user to input any URL and receive a class prediction. This can be used as a cybersecurity tool to help spread awareness and prevent attacks on users. Table 3 shows the implemented tool, which asks the user to input a URL to classify or to write "exit" to exit.

```
Type a URL to classify it.
Type 'exit' to stop.

Enter URL: www.syfy.com/rewind/?p=50206
Prediction: phishing

Enter URL: account-verification-chase.com/login/secure
Prediction: benign

Enter URL: http://www.nptw103.com.tw/news.php?news_id=6
Prediction: defacement

Enter URL: soft-downloads247.com/windows/update/install.exe
Prediction: malware

Enter URL: exit
Exiting.
```

Figure 3. Printed output of the prediction function.

V. CONCLUSION

This project explored the use of a lightweight transformer model, DistilBERT, to classify URLs as benign, phishing, malware, or defacement. By relying on raw URL text and avoiding heavy preprocessing, the model was able to learn meaningful structural and lexical patterns associated with malicious behavior. The training process was straightforward, and the model achieved strong performance, demonstrating that transformer-based approaches can effectively handle URL classification without relying on handcrafted features or external metadata.

The results show that DistilBERT is a practical and efficient option for real-world cybersecurity applications, especially in situations where speed and accuracy are both important. Its ability to generalize across different types of malicious URLs suggests that it can adapt to evolving threats, making it a strong candidate for integration into tools such as email filters, browser extensions, and automated URL-scanning systems.

ACKNOWLEDGMENT

We thank Dr. Mustafa Youldash for his guidance throughout this project. We would also like to acknowledge the creators of the Malicious URLs Dataset for making their work publicly available. Their contribution provided the foundation for our experiments and made this project possible.

REFERENCES

[1] P. Maneriker *et al.*, "URLTran: Improving Phishing URL Detection Using Transformers," *MILCOM 2021 – IEEE Military Communications Conference*, San Diego, CA, USA, 2021, pp. 197–204, doi: 10.1109/MILCOM52596.2021.9653028.

[2] A. El Mahdaouy *et al.*, "DomURLs_BERT: Pre-trained BERT-based model for malicious domains and URLs detection and classification," *arXiv preprint* arXiv:2409.09143, 2024.

[3] Sakshi Mahendru and T. Pandit, "SecureNet: A Comparative Study of DeBERTa and Large Language Models for Phishing Detection," pp. 160–169, Jul. 2024, doi: https://doi.org/10.1109/bdai62182.2024.10692765.

[4] Sultan Asiri, Y. Xiao, and T. Li, "PhishTransformer: A Novel Approach to Detect Phishing Attacks Using URL Collection and Transformer," *Electronics*, vol. 13, no. 1, pp. 30–30, Dec. 2023, doi: https://doi.org/10.3390/electronics13010030.

[5] Adel, H., Dahou, A., Mabrouk, A., & Ali, A. A., "The DistilBERT model architecture and components," in Improving Crisis Events Detection Using DistilBERT with Hunger Games Search Algorithm, Mathematics, Jan. 2022. (*researchgate.net*)

[6] sid321axn, "Malicious URLs Dataset," Kaggle. [Online]. Available: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset. Accessed: 09-Dec-2025.