جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⴰⵢⵏ

AL AKHAWAYN UNIVERSITY

School of Science and Engineering

Leveraging AI to Enhance Email Security and Phishing Email Detection

Internship Report

August 2024

Abdelilah Nossair

Supervised by:

Maha Tazi, Chief of Staff and Senior Consultant

and

Dr. Amine Amar - SSE Supervisor (Al Akhawayn University)

Student conduct and copyright

The student agree that:

1. Permission has been obtained for any third party content (eg Data, illustrations, photographs, charts or maps).

2. The results described in this report have not previously been published

Student's name and signature:

Abdelilah NOSSAIR

Internship Report

Student Statement:

"I, Abdelilah Nossair, have applied ethics to the design process and in the selection of the final proposed design. And I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable."

Student's name

Approved by the Supervisor

Dr. Amine AMAR

ACKNOWLEDGEMENTS

I wish to extend my sincerest gratitude to all who, by every means possible, assisted in bringing this project to its successful conclusion.

I also wish to express my appreciation to my academic supervisor, Dr. Amine Amar, from Al Akhawayn University, for his invaluable guidance, constant support, and motivation during the completion period of this internship. His comments and expertise have formed an integral part in setting the trend of this project.

I am also deeply indebted to Maha Tazi, Chief of Staff and Senior Consultant at Deloitte Morocco Cyber Centre, who afforded me the opportunity to take part in this great innovative project in the dynamic and professional context of the festival. Her remarks and advice went a long way in my learning and understanding of cybersecurity practices.

I would like to extend my thanks to the entire Cyber Risk Advisory Team at DMCC for their collaborative spirit, willingness to share their knowledge, and for making my internship experience enriching and engaging. Their expertise and support were invaluable to understand cybersecurity practices.

Special thanks to my family and friends for the continuous support, encouragement, and comprehension that has been so important during this exhausting but rewarding process. Their confidence in my abilities has always become a stimulus for me.

I would also like to thank the open-source community for providing the needed datasets and tools for this project. Without them, it would have been quite a task to discuss AI and machine learning in cybersecurity in this paper. Thank you all for your effort and dedication.

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

ABSTRACT (ENGLISH)

The rapid growth of digital technologies has brought about significant advancements in various sectors of the global economy but has also led to an increase in cybercrime, particularly phishing attacks. Phishing, a method of deception where attackers trick individuals into revealing sensitive information, remains one of the most prevalent and damaging forms of cyber threats. This paper outlines the development of an AI-based phishing email detection system aimed at enhancing cybersecurity measures by reducing human error and increasing detection accuracy.

Conducted as part of an internship at Deloitte Morocco Cyber Centre (DMCC), the project involves leveraging machine learning and deep learning models to create an effective solution for phishing detection. Traditional phishing detection methods are limited by their dependence on predefined rules and human intervention, making them ineffective against sophisticated and evolving phishing tactics. The AI-based approach developed in this project is scalable, automated, and adaptable in real-time to emerging threats.

The project utilizes various machine learning models, including Random Forest, Gradient Boosting, Logistic Regression, and DistilBERT, a transformer-based deep learning model fine-tuned for natural language processing (NLP) tasks. DistilBERT, in particular, demonstrated outstanding performance in detecting phishing emails, achieving high accuracy, precision, and recall. The model was integrated into a user-friendly Chrome extension, allowing real-time phishing detection within popular email clients like Outlook and Gmail.

This paper provides a comprehensive overview of the project development process, the models employed, and the methodologies used, along with the significant learnings acquired during the internship. It also highlights the potential limitations of AI in phishing detection and outlines future work, such as continuous training with updated datasets, integration with other security systems, and expansion across multiple platforms to ensure broader accessibility and effectiveness in combating phishing attacks.

Keywords: Phishing Detection, Artificial Intelligence, Cybersecurity, Natural Language Processing.

RESUME (FRENCH)

La croissance rapide des technologies numériques a permis des avancées significatives dans divers secteurs de l'économie mondiale, mais a également entraîné une augmentation de la cybercriminalité, en particulier des attaques de phishing. Le phishing, une méthode de tromperie où les attaquants manipulent les individus pour qu'ils révèlent des informations sensibles, reste l'une des formes de cybermenaces les plus courantes et les plus dommageables. Cet article présente le développement d'un système de détection des emails de phishing basé sur l'intelligence artificielle, visant à renforcer les mesures de cybersécurité en réduisant les erreurs humaines et en augmentant la précision de détection.

Réalisé dans le cadre d'un stage au Deloitte Morocco Cyber Centre (DMCC), le projet implique l'utilisation de modèles d'apprentissage automatique et de deep learning pour créer une solution efficace de détection de phishing. Les méthodes traditionnelles de détection de phishing sont limitées par leur dépendance à des règles prédéfinies et à l'intervention humaine, ce qui les rend inefficaces face aux tactiques de phishing sophistiquées et en constante évolution. L'approche basée sur l'IA développée dans ce projet est évolutive, automatisée et adaptable en temps réel aux nouvelles menaces.

Le projet utilise divers modèles de Machine Learning, notamment Random Forest, Gradient Boosting, Régression Logistique, et DistilBERT, un modèle de deep learning basé sur des transformateurs et affiné pour des tâches de traitement du langage naturel (NLP). DistilBERT, en particulier, a démontré une performance exceptionnelle dans la détection des emails de phishing, atteignant une grande précision, exactitude et rappel. Le modèle a été intégré dans une extension Chrome conviviale, permettant une détection en temps réel des tentatives de phishing au sein des clients de messagerie populaires tels qu'Outlook et Gmail.

Ce document fournit une vue d'ensemble complète du processus de développement du projet, des modèles utilisés, et des méthodologies appliquées, ainsi que des enseignements significatifs acquis au cours du stage. Il met également en lumière les potentielles limites de l'IA dans la détection de phishing et décrit les travaux futurs, tels que la formation continue avec des jeux de données actualisés, l'intégration avec d'autres systèmes de sécurité et l'expansion sur plusieurs plateformes afin d'assurer une accessibilité et une efficacité accrues dans la lutte contre les attaques de phishing.

Mots clés: Détection de Phishing, Intelligence Artificielle, Cybersécurité, Langage Naturel.

1. Introduction

The surge of digital technologies has dramatically transformed most sectors of the global economy in the past decade. However, such a revolution is always to be taken with caution, as it mainly fueled a parallel increase in cybercrime, where hostile actors kept innovating more and more sophisticated strategies of capitalizing on vulnerabilities of digital systems. One of the most common and devastating among such threats is phishing. Phishing attacks focus on human deception and trick individuals into revealing sensitive information, like login credentials, personal information, and financial data. It is the human element that is called the weakest link in cybersecurity worldwide because of this.

These phishing emails are a critical threat to cyber security, meaning they result in major financial losses, data breaches, and personal information leakage. This is actually calculated at 1.2% per day; it means that around 3.4 billion phishing emails are sent out daily. Phishing attacks account for 41% of all security incidents, while human error contributes to 74% of the cases, according to the Verizon Data Breach Investigations Report (2023). Misidentification of a phishing email can result in heavy financial and reputation loss. The results of such phishing attacks alone are responsible for $17,700 per minute, while the frequency of such attacks increased by 173% over the last years (CSO Online, 2024).

This calls for advanced technological solutions, with a central role played by Artificial Intelligence (AI) in boosting cybersecurity measures. My internship at Deloitte Morocco Cyber Centre (DMCC), the first cyber center established by Deloitte in Africa, helped me contribute to the dynamic landscape of cybersecurity. Located in Casablanca, DMCC is a center of excellence in cybersecurity with services spreading across a wide range of sectors such as banking, insurance, telecommunications, and public enterprises. Under the academic guidance of my supervisor at Al Akhawayn University, Dr. Amine Amar, and supervised by Maha Tazi, Chief of Staff and Senior Consultant at DMCC, I undertook the innovative project described in this paper with the aim of developing means for AI to better detect phishing emails.

The major task of the project was to develop a very strong AI-based solution for phishing email detection with high accuracy and efficiency, thus reducing human errors that result in security breaches. The traditional methods, based on predefined rules and manual intervention in phishing detection, are quite impotent against newer, sophisticated attempts that change very quickly. An AI-based approach would be scalable, automated, and adaptable in real near time to new, emerging threats. This report brings out a comprehensive overview of my internship experience, from the

details of the project development process to, most importantly, the significant learning acquired through the internship period.

2. Internship Background

2.1. Deloitte Morocco Cyber Center

DMCC is the premier hub for cybersecurity expertise in the whole of Africa, operating within Deloitte's international network. DMCC is based in Casablanca, Morocco, and provides high-tech solutions in cybersecurity according to international standards. It specializes in such services as assessment of risks, realization of frameworks of security, and management of incidents at response. Its aim is to serve its diversified clientele in banking, insurance, telecommunications, and public enterprises. The DMCC mission is to provide assistance to organizations in managing cybersecurity risks along with business continuity and compliance.

As an intern at DMCC, I was exposed to a dynamic environment in which innovation, collaboration, and continuous learning are fostered. Being guided by my supervisors, I soon became introduced to the multifaceted world of cybersecurity and the strategic approaches taken up by DMCC against emerging threats. The onboarding process underwent a series of courses online, starting with topics in risk management, asset security, and frameworks for both: security or information risk management. These were the very bases of my knowledge to contribute effectively to DMCC's projects in cybersecurity.

2.2. Cyber Risk Advisory Team

DMCC is composed of a Cyber Risk Advisory Team which helps organizations strengthen their cybersecurity postures to grapple with these complicated challenges and build toward a more secure tomorrow. This team strives to provide assistance and consulting in different cyber risk management fields, from consulting and audit to implementation, but also, managing security services, and incident response management.

The Cyber Risk team is structured in nine key pillars targeting specific focuses of cybersecurity:

- **Cyber Strategy:** Aligning clients' cybersecurity strategies with their broader business objectives.
- **Identity & Access Management (IAM):** Secure management of digital identities and access controls.
- **Detect & Respond:** Building the cyber defense and incident handling capabilities that let you react to these threats rapidly.
- **Cloud Security**: Guarantee confidentiality and integrity in how business processes moved to

the cloud can be secured.

- **Data & Privacy:** All sensitive data will be treated with the due level of care to protect the personal and organizational information.
- **Application Security:** This is the security of applications from the development lifecycle.
- **Infrastructure Security:** Protecting the assets of a network and infrastructure from cyber threats.
- **Emerging Technologies:** Ensuring next-generation technologies including IoT and Industrial Control Systems (ICS).
- **Offensive Security:** Measure the organization's cyber resilience by getting a penetration test and an ethical hack.

Collaboration with the Cyber Risk team is a living example whereby I have learned the practical complexities of how cybersecurity operates throughout industries associating governance, policies, and people's awareness along with technological solutions.

2.3. Focus on Phishing Email Detection

While interning, I went through brainstorming sessions with my supervisors for projects that would add immense value to DMCC's cybersecurity offerings. We discussed the possibilities of Network Intrusion Detection Systems, Credit Card Fraud Detection, and Cyberbullying Detection. Lastly, we concentrated on the innovation of "Leveraging AI for Email Security and Phishing Email Detection," based on the rise in the level of need for a fully automatic, accurate, and fast detection anti-phishing solution that minimizes human errors and boosts organizational defense. The third pointer is the basic issue of human errors and vulnerabilities in humans themselves when attacked by phishing, which in some way necessitates a solution with less dependence on human judgment. My project aimed to come up with an AI-based model for phishing detection that will learn from massive datasets, identifying patterns or activities associated with phishing attacks and accommodating new threats in real-time. This is very relevant to Deloitte's strategic vision: complex problems in cyber security require equally innovative solutions.

3. Problem Statement

Phishing is one of the most common and damaging forms of cybercrime, among a multitude of cybersecurity threats that are growing more sophisticated as digital transformation continues apace. A usual phishing attack involves a misleading attempt to obtain sensitive information through electronic communications under the guise of a trustworthy entity. These attacks can target the human element within an organization, which is the weakest link in cybersecurity defenses, leading to data breaches, financial losses, or other forms of damage.

The most significant task would be the identification of phishing emails from a large dataset. Traditional phishing detection methods rely heavily on rule-based systems and manual reviews, which are inherently not effective in handling the dynamic and evolving nature of phishing attacks. They rely a lot on predefined patterns and human intervention, making them less effective against novel phishing attempts that deviate from known patterns. This has, over the years, left organizations in a race to their feet with nascent ingenuity in phishing techniques, making their defenses stringent enough to detect the wily tactics.

The general problem I sought to solve with my internship project was:

How Artificial Intelligence can be applied to craft a reliable, automated, and scalable solution for detecting phishing emails in order to decrease human error, increase the accuracy of detection, and improve the general cybersecurity posture of organizations?

The objective of this project was to design a phishing detection model based on AI, which could learn from various datasets, figure out the pattern of phishing, and adapt to real-time emerging threats. This system aims to go beyond the limits of current phishing detection systems, including their rigidity, high false positive rates, and poor performance in the detection of new phishing strategies.

This AI-based approach further augments the detection of phishing emails and supports Deloitte's mission to deliver advanced technology solutions for the complex cybersecurity problem landscape. When organizations incorporate AI for their phishing detection process, the ability to identify and respond quickly to threats increases, providing improved protection to sensitive information with a reduced potential for damage.

4. Project Specifications

This project, whose working title is "Leveraging AI for Better Cyber Security and Detection of Phishing Emails," was designed to come up with a robust automated solution to accurately and efficiently detect phishing emails. In this regard, the primary objective of applying AI and Machine learning (ML) techniques is to eliminate shortcomings that are often characterized by rule-based conventional mechanisms for phishing detection, which are generally weak in tackling sophisticated and dynamically changing phishing attacks. Below, the main specifications of the project were structured:

4.1. Objective

This project was about putting in place and designing an AI-based phishing email detection mechanism that can accurately detect phishing attempts from voluminous and assorted datasets.

The solution had to ensure fewer errors by humans, better accuracy in detection, and have a scalable approach toward detecting phishing threats with no retention of sensitive data. The system needs to:

- Automatically detect phishing emails by using AI models trained on broad datasets of both legitimate and malicious emails.
- Ensure privacy and data security by destroying emails immediately after processing to comply with privacy concerns and data protection legislation.
- Avoid false positives and negatives, which will help in identifying phishing emails effectively without a problem with the valid communications.

4.2. Scope

The project involved building a machine learning model to be deployed as a Chrome extension embedded in email clients for both Outlook and Gmail. This includes:

- **Data Collection and Preprocessing:** Creating a dataset of phishing and legitimate emails from various sources, including open-source datasets. In addition to that, simulated phishing emails were also generated for training.
- **Feature Engineering:** Identifying key attributes that distinguish phishing emails from legitimate ones using text analysis, natural language processing techniques, URL analysis, too good to be true checks, and metadata-based features.
- **Model Development:** Building and training various machine learning models, including logistic regression, random forests, gradient boosting classifier, and transformer-based deep learning model, i.e., DistilBERT.
- **Model Evaluation and Tuning:** Testing model performance using different metrics and optimizing them through hyperparameter tuning.
- **Deployment as a Chrome Extension:** Deploying the final model as a Chrome extension for both Gmail and Outlook, allowing users to detect phishing emails directly within their inbox, offering real-time protection.
- **Data Privacy and Security:** Ensuring the model complies with data privacy protocols by deleting emails immediately after prediction, safeguarding privacy and adhering to data protection policies.

4.3. Tools and Technologies

The following tools and technologies were used in implementing the phishing detection model:

- **Programming Languages:**
    - **Python:** The core language chosen for its robust libraries in data science, machine

learning, and NLP, such as Pandas, Scikit-Learn, TensorFlow, and PyTorch.

- **JavaScript:** Used to develop Chrome extension components, including background scripts, content scripts, the user interface, and to manage browser interactions.
- **HTML/CSS:** Utilized to design the user interface of the Chrome extension.

- **Machine Learning Frameworks:**
  - **Scikit-Learn:** Used for building classical machine learning models like logistic regression and random forests.
  - PyTorch and TensorFlow: Employed for deep learning models, in our case, for DistilBERT.
- **Natural Language Processing Libraries:**
  - **Hugging Face Transformers:** Used for transformer-based models like DistilBERT that fine-tuned for phishing detection tasks.
- **Data Visualization Tools:**
  - **Matplotlib and Seaborn:** Utilized for data visualization and exploratory data analysis.
- **Web Framework:**
  - **Flask:** A lightweight Python web framework used to create a RESTful API endpoint for serving the phishing detection model. Flask facilitates communication between the Chrome extension (client-side) and the machine learning models (server-side), allowing real-time prediction requests and responses.

4.4. Expected Outcomes

The expected outcomes of the project include:

- **Development of an Efficient AI-based Phishing Detection Model:** A model capable of detecting phishing emails with high accuracy, precision, and recall.
- **Integration Capability:** The solution should integrate smoothly with existing cybersecurity frameworks and infrastructures used by organizations.
- **Compliance with Data Privacy Standards:** The framework minimizes data misuse and adheres to privacy guidelines by deleting emails immediately after prediction.

5. STEEPLE Analysis

5.1. Social

One of the key features of this tool is its AI-based phishing detection, which provides real-time alerts about potential phishing attacks and their risks. This helps create a more cyber-aware society by enabling users to recognize phishing threats and understand the associated risks.

Additionally, it builds user trust and confidence in the security of their email communications, which is essential for both personal and professional interactions. Deploying the solution as a Chrome extension for both Outlook and Gmail ensure wide accessibility, promoting inclusivity by making effective cybersecurity tools available to even low-tech users.

5.2. Technological

Leveraging advanced AI and machine learning, including the DistilBERT deep learning model, achieves high accuracy in phishing detection, showcasing AI's potential to enhance cybersecurity. Integrating with popular email platforms like Outlook and Gmail ensures a seamless user experience while embedding sophisticated AI models within existing infrastructure.

Though continuous learning from new data isn't possible due to privacy concerns, the model can be periodically updated with new datasets. This adaptability ensures the tool remains relevant and effective against evolving phishing tactics and emerging threats.

5.3. Economic

From an economic point of view, an AI system implemented to eliminate phishing results in relatively lower costs. As we all know, manual review by security teams are very time-consuming and expensive protective measures, whereas if most of such detection work is done automatically, it reduces a huge amount straight out of total expenses. Organizations can also avoid the large monetary toll of successful phishing attacks.

This increases productivity due to the fact that there is a reduction in false positives and the decrease in frequency of phishing attacks, which allows employees to concentrate on their work without being side-tracked by security issues. Advanced AI-based email security helps secure those businesses' clients who care about cybersecurity, helping them stay ahead of their competitors.

5.4. Environmental

The phishing detection tool, once deployed as a Chrome extension, does not entail any physical infrastructure; therefore, it is environment-friendly and has a low carbon footprint.

5.5. Political

It supports national and international policies in cybersecurity, which are directed at combating cybercrime and improving data protection. By enhancing the capability of detecting phishing, the project is part of government initiatives in safeguarding digital communication and infrastructure.

5.6. Legal

Data privacy is a major part of this project. The proposed tool for phishing detection is in compliance with major data protection regulations, such as GDPR. By removing emails immediately after processing, it makes sure that data privacy is maintained at its maximum and lessens the occurrence of cases of data breach or misuse. Organizations deploying the tool should make sure all their legal requirements are met and then some through provided terms of service and by obtaining the right consents.

5.7. Ethical

The project puts a lot of emphasis on user privacy and data security by not storing the personal information of emails analyzed but rather dropping them on the spot; it thus portrays an ethical manner of handling data. Transparency and interpretability are one of the main considerations of our project, the users must be kept in the know-how of what their data will be used for, even if just for real-time analysis. Another key issue is that you need to train your AI model using diverse datasets so you can avoid bias for phishing detection. A biased model may lead to asymmetrical protection levels, or false positives affecting certain users and types of emails more frequently.

6. Internship Plan

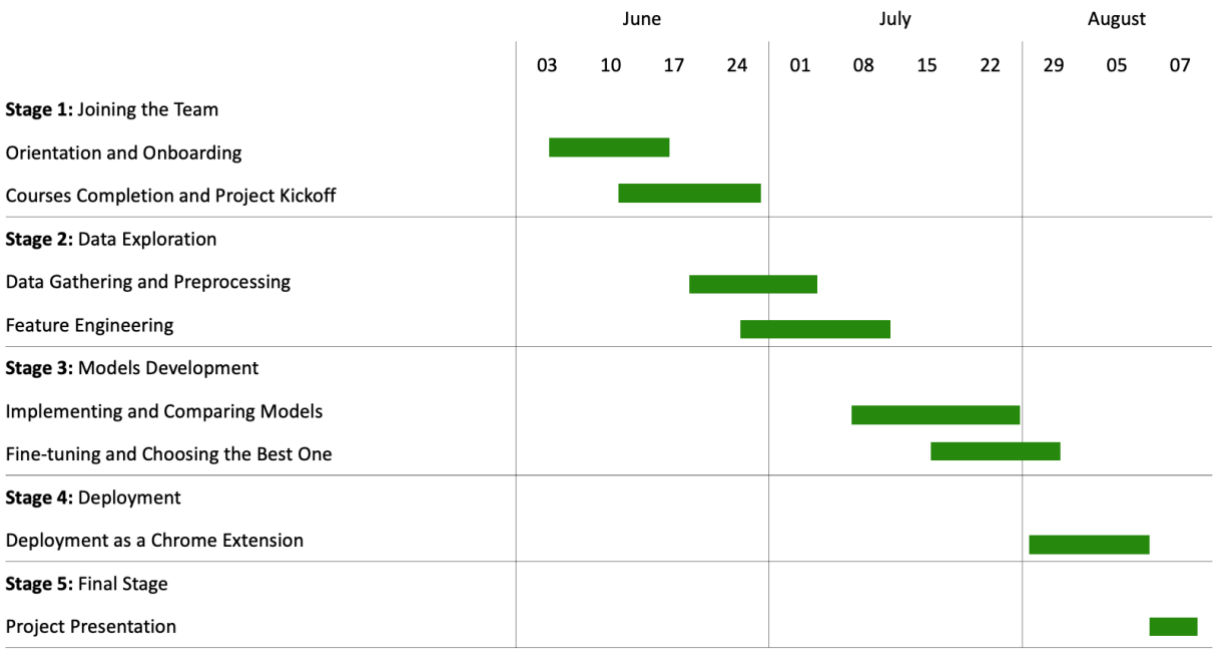


Figure 1: Gantt Chart for the Phishing Detection Project Internship Timeline

The internship was structured into five distinct stages, each with specific tasks and deliverables aimed at developing and deploying an AI-based phishing detection tool as a Chrome extension for Outlook and Gmail. The following is a detailed breakdown of each stage of the internship:

Stage 1: Joining the Team (June 3 - June 17)

- **Orientation and Onboarding (June 3 to June 10):** My first week was dedicated to the introduction to the team from Deloitte Morocco Cyber Center. This included getting to know the company's culture, DMCC cybersecurity policy, and the tools in use. The orientation took place, and core principles and practices in cybersecurity were briefed with an emphasis on one of the major concerns: phishing detection.
- **Courses Completion and Project Kickoff (June 10 - June 17):** After orientation, I passed through courses on cybersecurity and risk assessment and management as a build-up to my final project. After completion of the courses, a meeting for the kickoff of the project was set up, which highlighted objectives, scope, and what is expected of the final outcome by building the phishing detection model based on AI.

Stage 2: Data Exploration (June 17 - July 1)

- **Data Gathering and Preprocessing (June 17 - June 30):** From June 17 to June 30, we collected the related datasets for phishing detection. The dataset was composed of publicly available datasets and synthetic data. Data preprocessing tasks included cleaning, normalization, and removing duplicates, as well as handling missing values, to prepare the data for quality training of models.
- **Feature Engineering (June 24 - July 1):** Key features of the data were extracted to help the classifier distinguish phishing emails from legitimate ones using text analysis, URL, and too good to be true checks. This feature engineering played a critical role in making the model more accurate for phishing attempt detection.

Stage 3: Models Development (July 1 - July 22)

- **Training and Cross-Validation (July 1 – July 15):** This involved the development of many machine learning models and their training after having prepared the datasets. The implemented models included logistic regression, random forest, gradient boosting, and DistilBERT, and were compared using accuracy, precision, recall, and F1-score.

- **Model Fine-Tuning and Selection (July 8 – July 22):** After the initial models training, we fine-tuned them to get better performance. The set hyperparameters were adjusted so that the model detects phishing emails with the least number of false positives and negatives, leading to the selection of the best model for deployment.

Stage 4: Deployment (July 22 - August 5)

- **Deployment as a Chrome Extension (July 22 - August 5):** The selected model was deployed as a Chrome extension to work in the environment of Outlook and Gmail. A Flask app had been built, which uses a web-based approach to connect to the model's API endpoint for easier access. The extension was then tested for integration, usability, and performance to ensure seamless operation and real-time phishing detection within email platforms.

Stage 5: Final Stage (August 5 - August 7)

- **Project Presentation (August 5 - August 7):** The last step comprised of developing and presenting the results of the project to the team at DMCC and the supervisors. Information presented in the final presentation consisted of the background, objectives, methods, key findings, limitations, and recommendations for the future. Discussions obtained from the presentation were used to outline potential next steps for development.

7. Literature Review

7.1. A Literature Review on Classification of Phishing Attacks

S. Chanti and T. Chithralekha, in their paper "A Literature Review on Classification of Phishing Attacks" presented an excellent overview of various types of phishing attacks, the detection methods, and challenges involved. They have classified phishing attacks depending on how they are given to a target, such as email phishing, spear phishing, smishing or SMS phishing, and vishing or voice phishing, explaining how attackers take advantage of human psychology in each case.

It ranges from the review of various machine learning techniques in detecting phishing, ranging from traditional models like Random Forest and K-Nearest Neighbors (KNN) to deep learning methods using Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). It enumerates that deep learning models catch complex patterns in phishing attacks more effectively compared to traditional models.

It also focuses on feature extraction and engineering, insisting that the most important methods consider features concerning URLs, domains, and email headers. Furthermore, it insists on the integration of NLP techniques such as Word Embeddings to complement phishing detection systems.

It also highlights some of the most important challenges facing phishing detection, such as evolving tactics, the requirement for large labeled datasets, and high false positive rates, which have raised the imperative for model updates. Besides that, this review has pointed out human factors such as error, a lack of awareness, or perhaps even cognitive biases in phishing susceptibility and has supported bringing technical solutions together with user education in order to strengthen phishing defenses.

7.2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

The paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", written by Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, introduces BERT; a revolutionary language model that advances NLP with the idea of bidirectional training. Unlike the previous versions, such as ELMo and OpenAI GPT, which scored better in one direction, BERT looks at text from both directions. This allows it to capture more context and achieve superior results in NLP tasks.

Key innovations of BERT include Masked Language Modeling, where random words are masked in a text and predicted based on both left and right context, and Next Sentence Prediction, which allows the model to grasp relationships between sentences. These techniques allow BERT to perform well on various NLP tasks, ranging from question answering to language inference.

The enthusiasm for fine-tuning in several NLP tasks was taken into consideration while designing the architecture of BERT which achieved impressive results on benchmarks like GLUE, SQuAD, and SWAG. Experiments demonstrated that BERT greatly outperformed other previously reporting models on many NLP tasks, establishing it as an extremely powerful tool to understand natural languages.

7.3. BERT: A Review of Applications in Natural Language Processing and Understanding

"BERT: A Review of Applications in Natural Language Processing and Understanding", conducted by M.V. Koroteev, provides a full overview of what BERT is and how this system changed the face of NLP. This review underlines peculiarities of BERT mechanisms: bidirectional training, applications across diverse NLP tasks, comparisons with other models that put BERT at the forefront.

While the previous models, such as Word2Vec and GloVe, could only capture the dynamic nature of context to a certain limit, BERT is notably a paradigmatic shift because of its introduction of the Transformer architecture, which learns context both ways. These are achieved through two important innovations: Masked Language Modeling, where some tokens are masked and predicted based on context, and Next Sentence Prediction, which helps model relationships between sentences. These techniques allow BERT to perform well in tasks requiring deep semantic understanding, such as question answering and language inference.

The BERT architecture is flexible; it can easily be fine-tuned for different NLP tasks with minimal modifications. This even extends to those related to text classification and annotation, quality assessment of generated text, where it sets new standards of performance with its variants like BERTScore and outperforms previous established metrics. Moreover, the generation of word embeddings in context rather than in a vacuum greatly helps BERT resolve ambiguities in words and hence arrive at more accurate and contextually appropriate understandings.

8. Methodology

8.1. Approaches

The approach followed in this work implies the use of techniques of artificial intelligence and machine learning for the development of an effective model of phishing detection. The main objective was the automation of the identification process of phishing emails with high accuracy by textual content both from the subject and the body of an email. It consisted of several key steps:

- **Collecting and Preparation of the Data:** We curated nine publicly available datasets of phishing and legitimate emails.
- **Exploratory Data Analysis (EDA):** EDA was done for each dataset to comprehend the

structure of data, find patterns and structures in construction, and build on it feature extraction. This included analysis of shape and statistics, missing values, class distribution, email length, subject line content, URLs, special characters, and keywords.

- **Feature Engineering:** Based on the understanding developed from EDA, relevant feature extraction was performed, for instance, email length, count of URLs, special character ratio, and too good to be true keywords. Further, these features will be utilized in training different machine learning models capable of classifying phishing and legitimate emails.
- **Selection of Model:** Different machine learning models have been implemented and evaluated on specific performance metrics, such as accuracy, precision, recall, and the F1-score. The best performing model was selected for deployment.
- **Extension Development:** To deploy our model, a Chrome extension is developed which will integrate well with popular email platforms, such as Outlook or Gmail. The solution will be developed as an extension with HTML, CSS, and JavaScript for developing the user interface. In this way, users will be allowed to have the system highlight, in real time, suspicious emails in their inbox. It sends a request to the backend API about classifying emails either as phishing or legitimate, therefore improving user experience through immediate feedback without leaving the email platform.

8.2. Model Development and Training

During my internship, we explored various models of machine learning for the development of an efficient phishing detection system. All nine different datasets were combined for training the models, since this would result in a good improvement in the generalization capability of the model. Multiple datasets can provide better coverage for every type of phishing tactic and normal pattern of email, which is important for the model to be robust and trustworthy.

The goal is to assess the performance of each model in telling phishing from legitimate emails. After this, the models would be further trained and validated using relevant metrics such as accuracy, precision, recall, and F1-score. The best performing model in this case would then be forwarded for deployment in the phishing detection system. In the following, we discuss each model used in this project, its methodology of implementation, and its adequacy in phishing detection.

8.2.1. Random Forest Classifier

The Random Forest Classifier works by ensemble learning, a process of combining predictions from multiple decision trees to yield yet a more accurate and robust prediction. That is, it builds a "forest" of decision trees where each one of the trees is trained on a random subset of data. The final prediction is therefore made by aggregating the predictions across all the individual trees through majority voting for classification tasks.

- **Implementation Details:**
    - The Random Forest model was implemented using the RandomForestClassifier from the scikit-learn library in Python.
    - Key hyperparameters, such as the number of trees (n_estimators), maximum depth of the trees (max_depth), and the number of features to consider for each split (max_features), were tuned using grid search cross-validation to find the optimal settings.
- **Advantages for Phishing Detection:**
    - Random Forest is robust to overfitting because of averaging throughout the multiple decision trees within it, and therefore will be applicable for a dataset which contains mixed phishing tactics.
    - It can handle both numerical and categorical data and ranks feature importance automatically, hence giving insight into which feature-email length, URL count, special character ratio-is most influential for phishing detection.

8.2.2. Gradient Boosting Classifier

The other ensemble learning method is the Gradient Boosting Classifier, which builds models one by one; each subsequent model works on correcting those previous mistakes. While Random Forest builds trees independently, Gradient Boosting focuses on optimizing a loss function to improve the accuracy of the prediction incrementally.

- **Implementation Details:**
    - The Gradient Boosting model was implemented using the GradientBoostingClassifier from the scikit-learn library.
    - Key hyperparameters such as the learning rate (learning_rate), the number of boosting stages (n_estimators), and the maximum depth of each tree (max_depth) were optimized using grid search cross-validation.

- **Advantages for Phishing Detection:**
    - Gradient Boosting is effective at handling complex datasets where the relationship between features and the target variable (phishing or legitimate) is non-linear.
    - It provides greater control over model tuning through parameters like learning rate and tree depth, which can help prevent overfitting and improve generalization.

8.2.3. Logistic Regression

Logistic Regression is one of the most common linear models used for binary classification problems. It outputs the probability of a particular instance belonging to a given class through a logistic function. Simpler than ensemble methods, logistic regression is a very powerful baseline modeling for a classification problem; it is particularly useful when the relation between features and the target variable is roughly linear.

- **Implementation Details:**
    - The Logistic Regression model was implemented using the LogisticRegression class from the scikit-learn library.
    - Key hyperparameters such as the regularization strength (C) and the type of regularization (L1 or L2) were tuned using grid search cross-validation to prevent overfitting.
- **Advantages for Phishing Detection:**
    - Logistic Regression is interpretable, allowing us to understand the contribution of each feature to the prediction. This is useful for understanding which characteristics (e.g., presence of certain keywords, email length) are most indicative of phishing.
    - It is computationally efficient and serves as a good baseline to compare against more complex models.

8.2.4. DistilBERT

DistilBERT is a transformer-based deep learning model derived from BERT. It is a smaller, faster, and more efficient compared to BERT. It has been designed for NLP tasks such as text classification, sentiment analysis, and, as will be seen here, phishing detection. DistilBERT leans on the power of transformers for contextual understanding within emails, thus making them quite effective at email content analysis.

With BERT's bidirectional encoding, it reads the text both from left to right and right to left at the same time; hence, it understands the context of every word based on the other words surrounding it. That allows for a deep understanding of the patterns in language and semantics. DistilBERT retains that strong bidirectional understanding of context while being lightweight, hence effective in analyzing email content for subtle phishing indicators.

- **Implementation Details:**
    - The DistilBERT model was implemented using the transformers library from Hugging Face.
    - The model was fine-tuned on the combined dataset using transfer learning, where the pre-trained DistilBERT model was further trained on the email data specific to this task.
    - Due to the computational complexity of training transformer models, the training process was conducted using GPU acceleration to reduce time and improve efficiency.
- **Advantages for Phishing Detection:**
    - DistilBERT captures the contextual meaning of words and sentences, making it highly effective for understanding the nuances in email content that might indicate phishing.
    - The model can handle large text sequences and leverage pre-trained language understanding, making it highly adaptable to different types of phishing emails.
    - It is particularly suited for NLP-based phishing detection, where understanding the semantics and context of the email body and subject is crucial.

8.3. Application Development

The phishing detection system was designed to analyze emails on popular email platforms like Outlook and Gmail in real time. For this purpose, two major components were developed: a Flask API for the backend machine learning model inference and a Chrome Extension for the client-side user interaction.

8.3.1. Flask API

The backend of the phishing detection system was built using a Flask API, which serves as the interface between the machine learning model and the Chrome extension. The API is responsible

for receiving email content from the Chrome extension, processing it through the best trained model, and returning the probability of the email being phishing.

Key Components of the Flask API:

- ***Model Integration:***
    - The Flask API loads the selected model, fine-tuned over phishing detection, combining its prediction output with additional features from the email content, like urgent keywords, bad grammar, and suspicious attachments.
    - The model is loaded from a pre-trained checkpoint and set to evaluation mode to ensure that it is ready for inference.
- ***Feature Extraction Functions:***
    - A variety of helper functions are defined to carve out meaningful features in the body of the email. These include checks for "too good to be true" offers, checks for the presence of HTML tags, counting URLs, urgent action phrase check, and special character ratio analysis.
- ***Prediction Function:***
    - The predict function tokenizes the email content and extracts additional features. It then feeds the processed input into the model to calculate the probabilities with the softmax function.
    - The output probability of the email being a phishing one is calculated and returned to the client.
- ***API Endpoint:***
    - The API provides a "/detect" endpoint that listens for a POST request containing email content. After pre-processing, the model inference is called in the endpoint, which returns the probability of phishing in JSON format.
    - CORS (Cross-Origin Resource Sharing) is enabled so that requests from the Chrome extension are able to securely communicate with the backend API.
- ***Testing and Local Development:***
    - The Flask application was checked locally in the development environment for any functionality or performance issues and the compatibility of the Chrome extension.

8.3.2. Chrome Extension

The Chrome Extension provides the client-side interface through which the end user will access and interact with the phishing detection system. It is integrated within Outlook and Gmail, providing an on-the-fly phishing analysis for incoming emails.

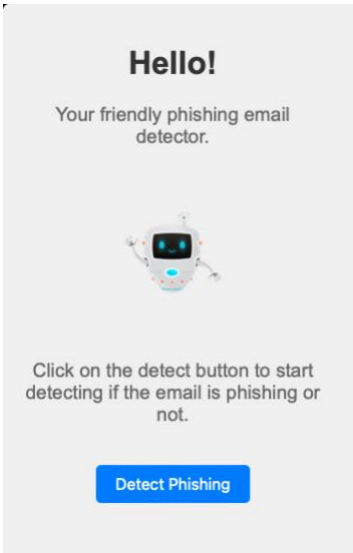


Figure 2: Phishing Email Detection Chrome Extension User Interface

Key Features of the Chrome Extension:

- ***User Interface:***
    - The extension is built using HTML, CSS, and JavaScript, providing a front-end user interface that is simple to use and integrates directly into the user interface of the emails' platform.
    - When an email is opened by the user, the extension sends the content of the email (Subject and Body) automatically to the Flask API to analyze it and shows the probability of phishing to the user in a very nonobtrusive manner.
- ***Real-Time Email Analysis:***
    - The extension listens for user action and wait for the "Detect Phishing" button to be triggered.
    - The content is sent as a JSON payload to the Flask API endpoint "/detect" for phishing detection.
- ***Handling API Responses:***
    - Upon receiving a response from the Flask API, the extension alerts the email recipient by showing the probability.

- If an email is flagged as potentially phishing, the user can take further action such as marking it as spam, deleting, or reporting.

- ***Security and Privacy:***
  - The extension guarantees that no sensitive data is stored or retained, once analyzed. All the processing will be done in real time, and the emails will be discarded right after receiving the response from the API, with no sensitive data maintained to override any privacy regulations.
- ***Development and Testing:***
  - The extension has been developed and tested in the Chrome browser to ensure that everything will work as smoothly as possible.

9. Data Presentation

9.1. Loading the Datasets

In this project, we developed a comprehensive AI-powered phishing detection model using datasets from various sources, containing both phishing and legitimate emails. These diverse datasets capture a wide range of real phishing attempts and legitimate communications, ensuring the model can generalize well across different phishing tactics.

These datasets were obtained for this project from two major repositories:

1. Curated Phishing Email Dataset (Source: Figshare)

This dataset is contributed by A. I. Champa, M. F. Rabbi and M. F. Zibran and the paper describing this dataset is "Curated datasets and feature analysis for phishing email detection with machine learning", in 3rd IEEE International Conference on Computing and Machine Intelligence (ICMI), 2024.

The dataset consists of seven repositories. Two, Ling and Enron, consist of only two features: 'Subject' and 'Body'. The rest of the datasets are comprised of six features: 'Sender', 'Receiver', 'Date', 'Subject', 'Body', and 'Urls'.

2. Nazario and Nigerian Fraud Datasets (Source: Kaggle)

These are taken from Kaggle: https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset. The emails were actually collected from phishing campaigns and contain spam and fraudulent emails by Nazario, and Nigerian scam emails from Nigeria Fraud.

The features included in these datasets are six in number: 'Sender', 'Receiver', 'Date', 'Subject', 'Body', and 'Urls'. These datasets provide further examples of phishing attempts, and the inclusion of which would be beneficial to enhance the robust model for phishing detection.

Feature	Description	Data Type
Sender	The email address of the sender who sent the email.	String
Receiver	The email address of the recipient(s) to whom the email is sent.	String
Date	The date and time when the email was sent.	DateTime
Subject	The subject line of the email, usually a brief summary of the email's content.	String
Body	The main content of the email, which can include text, images, links, and attachments.	String
Urls	The URLs or their count included in the body of the email, which may lead to external sites or resources.	String/ Integer

Table 1: Curated Metadata Fields, Descriptions, and Data Types

9.2. Exploratory Data Analysis (EDA)

EDA is one of the most important steps in data analysis, enabling to understand characteristics such as the structure, distributional characteristics, and variables of data that have been or will be used for a specific machine learning model. In this project, EDA was performed on each of the datasets separately to enable us to comprehend, visualize patterns, and identify anomalies in the

data as a preliminary to feature extraction. The most salient aspects of the EDA performed included analyzing dataset shape, summary statistics, checking for missing values, class distribution, length of emails, subject line analysis, URLs and special characters, and keyword analysis.

EDA Steps and Findings:

9.2.1. Dataset Overview: Shape, Summary Statistics, Missing Values, and Data Types

- For each dataset, the initial preprocessing included a shape check to know how many rows and columns were present, hence giving an overview of the size and structure of the dataset.
- Summary statistics, such as describe(), were employed to understand the mean and median, and standard deviation of numeric features such as email length and URL count. It was helpful in getting an intuition about possibly existing outliers and about the range of values.
- Identified missing values using isnull().sum() in order to get the count of missing entries per feature. For data quality and consistency, handling missing values is an important aspect. Missing data were removed.
- Checked the datatypes of each feature to ensure appropriate data types for further analysis and training of models.

```
Shape of the SpamAssasin dataset: (5809, 7)
Summary Statistics:
             label         urls
count  5809.000000  5809.000000
mean      0.295748     0.861078
std       0.456418     0.345895
min       0.000000     0.000000
25%       0.000000     1.000000
50%       0.000000     1.000000
75%       1.000000     1.000000
max       1.000000     1.000000
Missing Values:
sender        0
receiver    210
date          0
subject      16
body          1
label         0
urls          0
dtype: int64
Data Types:
sender      object
receiver    object
date        object
subject     object
body        object
label        int64
urls         int64
dtype: object
```

Figure 3 : Shape, Summary Statistics, Missing Values, and Data Types of SpamAssassin Dataset

9.2.2. Class Distribution Analysis

- Histograms were used to visualize the class distribution, showing the number of legitimate versus phishing e-mails for each dataset. This served to highlight if there were class imbalance problems, which intrinsically introduce bias into model predictions. Class balancing methods were considered (under-sampling) to deal with data unbalancing.

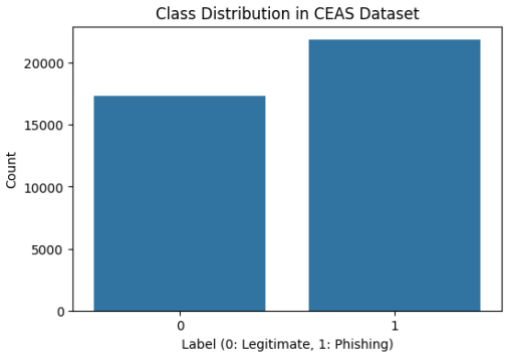


Figure 4 : Class Distribution of CEAS Dataset

9.2.3. Email Length

- The 'Subject' and 'Body' columns were combined in order to get a length of each email. In addition to that, the comparison in the distribution of length between phishing and legitimate emails was done using histograms.
- It was observed from the analysis that phishing emails vary in length compared to a valid email. This provides a very important insight into feature engineering; the length of an email is a good candidate to tell a phishing one apart.

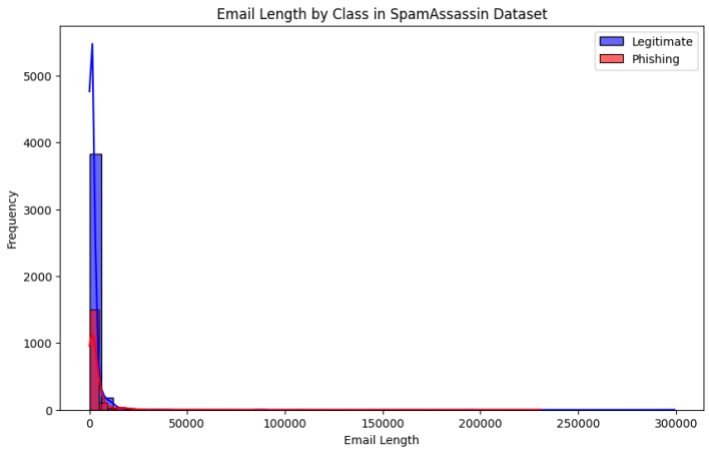


Figure 5 : Email Length by Class (Phishing and Legitimate) of SpamAssassin Dataset

9.2.4. Subject Line Analysis

- Analysis of email subject lines was conducted to figure out common patterns that frequently occur in phishing emails versus legitimate ones. Word clouds were generated to show the most common words within the subject lines for both phishing and legitimate emails.
- Separate word clouds were made for both phishing and legitimate emails to show the differences in the use of keywords and the kind of language used in subject lines. In fact, most phishing emails contain urgent or enticing languages to deceive recipients, while other legitimate emails use neutral language.

Figure 6 : Word Clouds for Legitimate and Phishing Subject Lines of Enron Dataset

9.2.5. URL and Special Character Analysis

- Analyzing of visualizations by using histograms and box plots that show the distribution of URL counts in phishing and legitimate emails.
- The special character ratio was calculated, denoting the ratio of special characters (like !, @, #, …) by the total length of the email body. This feature is helpful because special characters are often used in phishing emails to mask URLs or texts and hence could be helpful in differentiating phishing.

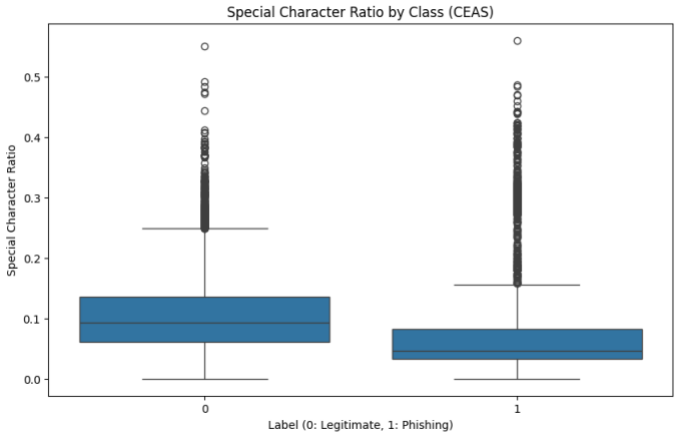


Figure 7 : Special Character Ratio for Legitimate and Phishing Subject Lines of CEAS

9.2.6. Keyword Analysis

- In the email body, a keyword analysis was performed to identify any of those "urgent" keywords commonly used in phishing emails. Keywords such as "urgent," "immediately," "action required," and "important update" were identified and counted in each email.

General Findings from EDA:

- Various data analyses were done on each of these datasets, and they proved to be much informative with regards to the structure and pattern that both phishing and legitimate emails take.
- To seek the significant difference between phishing and legitimate emails, several factors were taken in by email length, subject line content, count of URLs and special characters, and presence of urgent keywords.
- These findings provided the very basis for feature extraction; these detected patterns and characteristics represent the direct specifications of which features would be most useful to train a model on and predict.

10. Simulations, Results, and Interpretation

10.1. Simulations

The simulations involved training and evaluating four different models—Random Forest, Gradient Boosting, Logistic Regression, and DistilBERT—to detect phishing emails. The goal was

to assess each model's performance and select the best one for deployment in a real-time Chrome extension. The following steps were undertaken:

10.1.1. Data Preparation

- Datasets were split into training, validation, and test sets. The training set was used for model building, the validation set to control overfitting, and the test set for final evaluation.
- Features extracted during the EDA, such as email length, URL count, special character ratio, presence of specific keywords, and more, were used to train the traditional machine learning models (Random Forest, Gradient Boosting, and Logistic Regression).
- DistilBERT utilized raw email content (subject and body) in addition to extracted features and was fine-tuned on the curated datasets to understand contextual information within emails.

10.1.2. Initial Training

- All models were initially trained using default hyperparameters to establish baseline performances.
- Random Forest and Gradient Boosting models were trained using ensemble techniques to capture feature importance and interaction.
- Logistic Regression served as a baseline linear model, offering interpretable results.
- DistilBERT was fine-tuned on the emails content in addition to the concatenation with the extracted features which was passed through a classification layer to make predictions.

10.1.3. Hyperparameter Tuning

a. Random Forest

The Random Forest model was tuned to improve its performance by adjusting parameters that control the model's complexity, tree structure, and training procedure.

- **Parameter Grid:**
    - **n_estimators:** Number of trees in the forest (100, 200, 300).
    - **max_features:** Number of features to consider when looking for the best split ('auto', 'sqrt', 'log2').
    - **max_depth:** Maximum depth of each tree (None, 10, 20, 30).

- **min_samples_split:** Minimum number of samples required to split a node (2, 5, 10).
- **min_samples_leaf:** Minimum number of samples required to be at a leaf node (1, 2, 4).
- **bootstrap:** Whether bootstrap samples are used when building trees (True, False).

- **Randomized Search Setup:**
  - RandomizedSearchCV was used with 50 iterations, 3-fold cross-validation, and all available CPU cores.
  - The best hyperparameters were selected based on the highest cross-validation performance.
- **Best Parameters Found:**
  - {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 20, 'bootstrap': True})

b. Gradient Boosting

The Gradient Boosting model was fine-tuned by optimizing its parameters that control the learning rate, the complexity of individual trees, and the number of boosting stages.

- **Parameter Grid:**
  - **n_estimators:** Number of boosting stages (100, 200, 300, 400, 500).
  - **learning_rate:** Step size shrinkage used to prevent overfitting (0.01, 0.05, 0.1, 0.2).
  - **max_depth:** Maximum depth of individual trees (3, 4, 5, 6, 7).
  - **min_samples_split:** Minimum number of samples required to split a node (2, 5, 10).
  - **min_samples_leaf:** Minimum number of samples required to be at a leaf node (1, 2, 4).
- **Randomized Search Setup:**
  - RandomizedSearchCV was used with 50 iterations, 3-fold cross-validation, and all available CPU cores.
  - The best hyperparameters were selected to maximize performance metrics during cross-validation.
- **Best Parameters Found:**
  - {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': 7, 'learning_rate': 0.1}

c. Logistic Regression

The Logistic Regression model was tuned by optimizing its regularization and solver parameters, which impact the model's decision boundary and convergence.

- **Parameter Grid:**
  - **C:** Inverse of regularization strength (0.01, 0.1, 1, 10, 100).
  - **penalty:** Type of regularization to be applied ('l1', 'l2').
  - **solver:** Algorithm to use in optimization ('liblinear', 'saga').
- **Randomized Search Setup:**
  - RandomizedSearchCV was used with 50 iterations, 3-fold cross-validation, and all available CPU cores.
  - The best hyperparameters were identified based on performance metrics during cross-validation.
- **Best Parameters Found:**
  - {'solver': 'liblinear', 'penalty': 'l2', 'C': 100}

10.2. Results

After training and tuning, each model was evaluated on the test set using various performance metrics. These metrics included Accuracy, Precision, Recall, and F1-score. The results are summarized as follows:

a. Before Training

	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Random Forest	84.56	85.18	81.56	83.33
Gradient Boosting	77.99	76.91	76.43	76.67
Logistic Regression	64.79	64.74	56.12	60.13

Table 2 : Performance Metrics Before Tuning of Random Forest, Logistic Regression, and Gradient Boosting

b. After Training

	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Random Forest	84.69	85.23	82.17	83.67
Gradient Boosting	83.56	83.94	80.69	82.28
Logistic Regression	64.85	64.79	56.30	60.25

Table 3 : Performance Metrics After Tuning of Random Forest, Logistic Regression, and Gradient Boosting

c. DistilBERT

	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
DistilBERT	98.08	98.05	98.10	98.07

Table 4 : Performance Metrics for DistilBERT

10.3. Interpretations of Results

After training and fine-tuning each model—Random Forest, Gradient Boosting, Logistic Regression, and DistilBERT—on the phishing detection task, we analyzed their performance using various metrics such as Accuracy, Precision, Recall, and F1-score. The models were evaluated on a test set to ensure their ability to generalize to unseen data.

1. Logistic Regression

- **Low Performance:** Logistic Regression had the lowest performance among the models tested. Even after hyperparameter tuning, the improvements were minimal. This can be attributed to the linear nature of Logistic Regression, which makes it less suitable for complex patterns in data, such as those found in phishing emails.
- **PCA Analysis:** The PCA plot shows that the data points of different classes are clustered together with no clear separation. This indicates that the data is not linearly separable, which directly impacts Logistic Regression's ability to distinguish between phishing and legitimate emails.

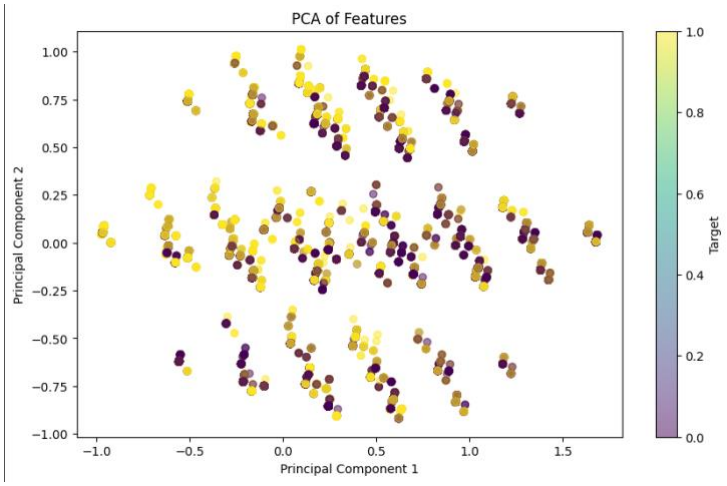


Figure 8 : Principal Component Analysis of features

- **Conclusion:** Logistic Regression's low F1-score indicates a high rate of both false positives and false negatives, making it an unreliable model for this task. It serves as a good baseline but is not ideal for deployment in a real-world phishing detection system.

2. Random Forest

- **Moderate Performance:** Random Forest showed better performance compared to Logistic Regression, with an accuracy of 84.69% after tuning. This indicates that it can handle non-linear relationships better due to its ensemble nature.
- **Minor Improvement with Tuning:** Hyperparameter tuning resulted in slight improvements in accuracy, precision, recall, and F1-score, indicating that the model's performance was close to optimal even before tuning.
- **Feature Importance:** Random Forest naturally provides feature importance, helping in understanding which features contribute the most to its predictions. Features like special_char_ratio and url_count were among the most significant.

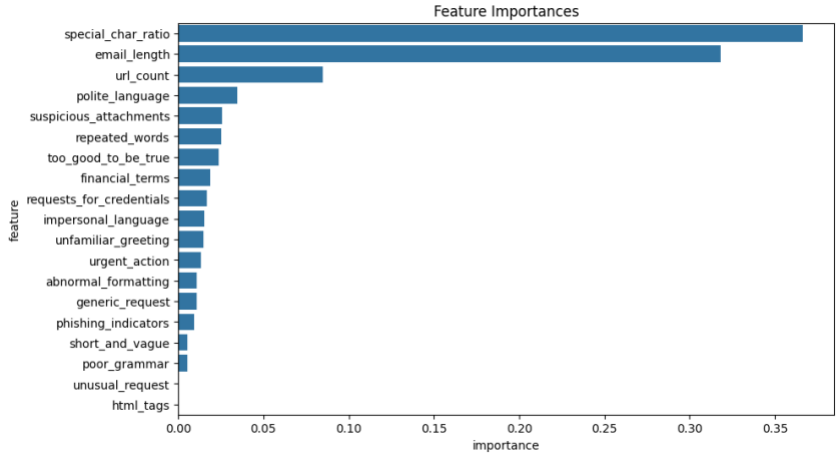


Figure 9 : Feature Importance for Random Forest Classifier

- **Conclusion:** Random Forest offers a good balance between performance and interpretability. However, its overall performance, though better than Logistic Regression, does not match that of more advanced models like Gradient Boosting and DistilBERT.

3. Gradient Boosting

- **Good Performance and Improved Results:** Gradient Boosting showed notable improvement after tuning, with an increase in accuracy from 77.99% to 83.56%. This model benefits from its ability to optimize errors of weak learners sequentially.
- **Handling Complex Patterns:** Gradient Boosting's improved performance after tuning indicates that it is well-suited to capturing more complex interactions between features, making it a powerful tool for phishing detection.
- **Versatility in Feature Interaction:** Gradient Boosting handles various feature types well, and its boosted decision trees can effectively differentiate between subtle phishing attempts and legitimate emails.

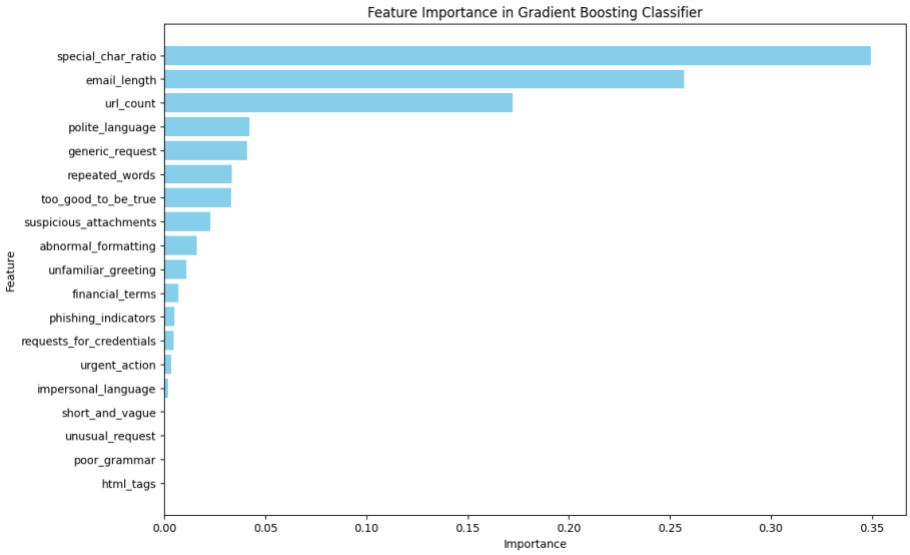


Figure 10 : Feature Importance for Gradient Boosting Classifier

- **Conclusion:** While Gradient Boosting outperforms Logistic Regression and is comparable to Random Forest, it still falls short compared to DistilBERT, particularly in recall and precision, which are crucial for minimizing missed phishing attacks.

4. DistilBERT

- **Outstanding Performance:** DistilBERT, fine-tuned with extracted features, achieved the highest performance across all models, with an accuracy of 98.08% and an F1-score of 98.07%. The combination of raw text analysis and feature extraction provided a comprehensive approach to detecting phishing emails.
- **Effective Handling of Contextual Information:** DistilBERT's ability to understand contextual nuances within the text allowed it to outperform other models that relied solely on engineered features.
- **Low False Positives and Negatives:** The high precision and recall rates indicate that DistilBERT has a low rate of both false positives and false negatives, making it highly reliable for phishing detection.
- **Hybrid Approach Advantage**: The addition of extracted features to the deep learning embeddings enhanced the model's ability to capture both linguistic subtleties and feature-based patterns associated with phishing.
- **Conclusion:** DistilBERT's superior performance in all key metrics makes it the best choice for deployment in a real-time phishing detection system. Its deep learning architecture combined with feature extraction provides a robust solution to the problem.

11.Learning Strategies

The successful completion of this project required a well-structured learning approach to understand the intricacies of machine learning, deep learning, and cybersecurity concepts involved in phishing detection. The following learning strategies were adopted:

1. Foundational Understanding of Phishing and Cybersecurity:

- **Domain Expertise Development:**
    - Engaged in discussions with cybersecurity professionals to gain practical insights into the current landscape of phishing attacks and defense mechanisms.
    - Participated in cybersecurity webinars and online courses to build a foundational understanding of how phishing emails are crafted and delivered.
- **Research and Literature Review:**
    - Conducted a literature review on phishing attacks and cybersecurity threats. This involved studying various types of phishing attacks, understanding their characteristics, and how they exploit human vulnerabilities.

2. Technical Skill Development in Machine Learning and Deep Learning:

- **Understanding Machine Learning Models:**
    - Studied various machine learning algorithms (Random Forest, Gradient Boosting, Logistic Regression) to understand their underlying mechanics, strengths, and limitations in handling phishing detection tasks.
    - Focused on learning the differences between ensemble models like Random Forest and Gradient Boosting and linear models like Logistic Regression. This helped in understanding the importance of model choice based on the complexity of the data.
- **Deep Learning with NLP Focus:**
    - Spent considerable time understanding Natural Language Processing (NLP) techniques and transformer-based models, such as DistilBERT. This included learning about transformers' attention mechanisms, how they handle contextual data, BERT bidirectional encoding, and their effectiveness in text classification tasks.
    - Worked on tutorials and documentation related to Hugging Face Transformers and PyTorch to implement and fine-tune DistilBERT for phishing detection.

- Developed skills in feature extraction techniques to complement DistilBERT's capabilities by incorporating email-specific features (e.g., URL count, special character ratio) with deep learning embeddings.

3. Practical Application through Hands-On Experiments

- **Experimentation with Model Training and Evaluation:**
  - Performed hands-on experiments with different models, starting with Logistic Regression as a baseline and then advancing to more complex models like Random Forest, Gradient Boosting, and DistilBERT.
  - Developed an understanding of overfitting and underfitting by observing model performance on training and validation datasets and implemented cross-validation techniques to mitigate these issues.
  - Learned to fine-tune hyperparameters for each model using RandomizedSearchCV to achieve optimal performance, balancing bias and variance for effective phishing detection.
- **Data Preprocessing and Feature Engineering:**
  - Gained experience in data preprocessing steps, including data cleaning, handling missing values, and feature scaling. This was crucial in preparing the data for training traditional machine learning models.
  - Focused on feature engineering to create meaningful input features that could help improve the models' ability to distinguish between phishing and legitimate emails.
- **Visualization and Interpretation:**
  - Used data visualization libraries like Matplotlib and Seaborn to conduct EDA and to understand the distribution and relationship between different features.
  - Interpreted results using confusion matrices, performance metrics (accuracy, recall, precision, F1-score) and feature importance plots to make informed decisions about model selection and further improvements.

4. Integration and Deployment Skills

- **API Development and Integration:**
  - Learned to build a backend API using Flask to serve the trained DistilBERT model and integrate it with a real-time Chrome extension for phishing detection.

- Acquired knowledge in RESTful web-services principles and CORS to ensure seamless communication between the frontend (Chrome extension) and backend (Flask API).

- **Chrome Extension Development:**
  - Developed skills in JavaScript, HTML, and CSS to build a user-friendly Chrome extension that could provide real-time phishing detection alerts to users.
  - Understood the challenges involved in developing machine learning models to client-side applications and optimizing them for performance and security.

12. Conclusion

The development of the AI-based phishing detection system at Deloitte Morocco Cyber Centre was holistic and enriching, combining the latest in machine learning and deep learning with real-world practical application in cybersecurity. The solution focused on developing a robust, scalable, and automated means of detecting phishing emails with high accuracy, reducing human error and contributing to organizational defenses.

It was a multistage project; from profound investigation of the problem domain, that is, the recognition of the urgency of phishing attack counteracting, one of the most topical kinds of cyber threats, to design of advanced solution to avoid dynamic exploitation of human vulnerabilities. The traditional rule-based detection methods cannot cope with sophisticated strategies of cybercriminals. This underlined the requirement to adopt AI and ML technologies that can learn from data, cope with new patterns, and protect in real time.

To this effect, it was a diversified collection of datasets, comprised of phishing and legitimate emails coming from different repositories, that formed a strong basis in training and testing the various models. These included the Random Forest, Gradient Boosting, Logistic Regression, and DistilBERT. The performance of each model needed to be rigorously analyzed with respect to standard evaluation metrics such as accuracy, precision, recall, and F1-score. These sets of experiments underlined the fact that traditional models, such as logistic regression, were a good baseline, but ensemble methods, in our case, Random Forest and Gradient Boosting, improved the detection rates. The deep learning model chosen, DistilBERT, seemed to perform outstandingly. DistilBERT fitted on the contents of emails and with the addition of features extracted from emails performed better regarding accuracy, yielding the best performance metrics; therefore, it was selected for development due to its exceptional ability to capture subtlety in language and context expressed in e-mails.

The model is operationalized by developing a functional, user-friendly Chrome extension that works seamlessly with famous email clients like Outlook and Gmail. It leverages the backend through a Flask API that supports real-time phishing detection, by which the extension can interface with a machine learning model to provide immediate alerts to the end-user. This streamlines a major step toward implementing AI-based phishing detection from research into a deployable solution that can protect the average user of email against cyber threats.

Throughout this internship, several key learning strategies were employed, encompassing specific knowledge in cybersecurity and phishing, acquisition of technical skills in machine learning and deep learning, and hands-on experimenting with model development. This experience indeed brought great depth in understanding the intricacies involved in crafting robust cybersecurity solutions. Moreover, it bestowed hands-on experience in integrating AI models into real-world applications, emphasizing data privacy among other ethical considerations.

While this AI-based phishing detection system means significant strides for the cybersecurity community, it does inherently house limiting factors. AI is indeed a double-edged sword. The very same AI that empowers improvements in phishing detection can be used by an attacker to generate intelligent phishing emails that will evade detection. Moreover, the actual performance will heavily depend on how representative the training datasets are. If the training data does not represent cases of phishing, it might be the case that the model will fail in recognizing some new, vague, or unique tactics of phishing, either as a false positive or a false negative.

Future improvements focus on enhancing the system’s adaptability and reach. Implementing continuous training methods that ensure the model stays updated with new phishing tactics, while protecting user privacy, which by all means is the priority. Furthermore, integrating the phishing detection tool with other security systems, like spam filters and malware detection, could provide a multi-layered defense strategy. Expanding cross-platform compatibility to support all major email providers and browsers would also widen accessibility, making the tool a universal solution for phishing protection.

In conclusion, the project demonstrated AI capability for the enhancement of phishing detection, thus providing a scalable, adaptable, and effective solution that would enable organizations to further harden their cybersecurity defenses. Developing the Chrome extension as a protection tool within real-time reality is a concrete step that aligns with Deloitte's mission to deliver innovative solutions for complex cybersecurity challenges. However, the use of AI is a two-edged sword; as much as it can improve detection, it is also used by the attackers in building more sophisticated phishing emails. In future research, fine-tuning of the model with fresh data will be done to keep it ahead of these threats, personal data privacy while performing the model updating, integration of the proposed detector in other security systems to make

up for an integrated defense strategy, and enhancement of cross-platform support in the interest of universal running. These efforts will ensure that the solution remains robust and effective against the constantly evolving landscape of phishing attacks, further solidifying its contribution to the field of cybersecurity.

References (APA)

2024 Data Breach Investigations report. (n.d.). Verizon Business. https://www.verizon.com/business/resources/reports/dbir/

Champa, A. I., Rabbi, M. F., & Zibran, M. F. (2024). Curated Datasets and Feature Analysis for Phishing Email Detection with Machine Learning. *3rd IEEE International Conference on Computing and Machine Intelligence*. https://doi.org/10.1109/icmi60790.2024.10585821

Chanti, S., & Chithralekha, T. (2022). A literature review on classification of phishing attacks. *International Journal of Advanced Technology and Engineering Exploration*, *9*(89). https://doi.org/10.19101/ijatee.2021.875031

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018, October 11). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv.org. https://arxiv.org/abs/1810.04805

Koroteev, M. V. (2021). BERT: A review of Applications in Natural Language Processing and Understanding. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2103.11943

Phishing email dataset. (2024, May 24). Kaggle. https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset

Wong, C. (2024, February 20). *What is phishing? Examples, types, and techniques*. CSO Online. https://www.csoonline.com/article/514515/what-is-phishing-examples-types-and-techniques.html

APPENDIX :

Import Necessary Librairies

```
[ ] import pandas as pd
    from sklearn.utils import resample
    from sklearn.model_selection import train_test_split
    from transformers import DistilBertTokenizer, DistilBertModel, Trainer, TrainingArguments
    from torch.utils.data import Dataset
    import torch
    from sklearn.metrics import accuracy_score, precision_recall_fscore_support
    import numpy as np
    import re
    import torch.nn as nn
```

Load Datasets

```
[ ] ceas = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/CEAS_08.csv')
    spamassassin = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/SpamAssasin.csv')
    trec_1 = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/cleaned_TREC_06.csv')
    trec_2 = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/cleaned_TREC_05.csv')
    trec_3 = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/cleaned_TREC_07.csv')
    enron = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/Enron.csv')
    ling = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/Ling.csv')
    nazario = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/Nazario.csv')
    nigerian_fraud = pd.read_csv('/content/drive/MyDrive/Phishing_Email_Detection/Nigerian_Fraud.csv')
```

Exploratory Data Analysis (EDA)

EDA for CEAS Dataset

Display First Few Rows

```
[ ] print("CEAS Dataset Head:")
    print(ceas.head())
```

```
CEAS Dataset Head:
                                              sender  \
0                   Young Esposito <Young@iworld.de>
1                       Mok <ipline's1983@icable.ph>
2  Daily Top 10 <Karmandeep-opengevl@universalnet...
3                Michael Parker <ivgrnai@pobox.com>
4  Gretchen Suggs <externalsep1@loanofficertool.com>

                                        receiver  \
0                    user4@gvc.ceas-challenge.cc
1                   user2.2@gvc.ceas-challenge.cc
2                   user2.9@gvc.ceas-challenge.cc
3  SpamAssassin Dev <xrh@spamassassin.apache.org>
4                   user2.2@gvc.ceas-challenge.cc

                               date  \
0  Tue, 05 Aug 2008 16:31:02 -0700
1  Tue, 05 Aug 2008 18:31:03 -0500
2  Tue, 05 Aug 2008 20:28:00 -1200
3  Tue, 05 Aug 2008 17:31:20 -0600
4  Tue, 05 Aug 2008 19:31:21 -0400

                                      subject  \
0                       Never agree to be a loser
1                          Befriend Jenna Jameson
2                            CNN.com Daily Top 10
3  Re: svn commit: r619753 - in /spamassassin/tru...
4                       SpecialPricesPharmMoreinfo

                                                body  label  urls
0  Buck up, your troubles caused by small dimensi...      1     1
1  \nUpgrade your sex and pleasures with these te...      1     1
2  >+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+...      1     1
3  Would anyone object to removing .so from this ...      0     1
4  \nWelcomeFastShippingCustomerSupport\nhttp://7...      1     1
```

Shape, Statistics, Missing Values, and Data Types

```
[ ] print("Shape of the CEAS dataset:", ceas.shape)
```

```
Shape of the CEAS dataset: (39154, 7)
```

```
[ ] print("Summary Statistics:")
    print(ceas.describe())
```

```
Summary Statistics:
              label         urls
count  39154.000000  39154.00000
mean       0.557848      0.66997
std        0.496649      0.47023
min        0.000000      0.00000
25%        0.000000      0.00000
50%        1.000000      1.00000
75%        1.000000      1.00000
max        1.000000      1.00000
```

```
    print("Missing Values:")
    print(ceas.isnull().sum())
```

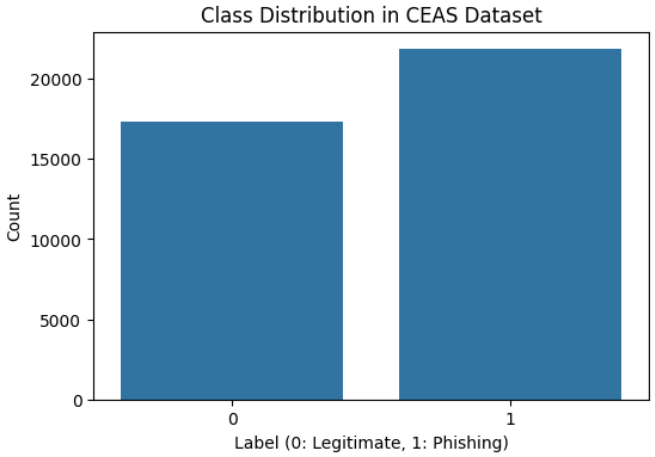
```
Missing Values:
sender        0
receiver    462
date          0
subject      28
body          0
label         0
urls          0
dtype: int64
```

```
[ ] print("Data Types:")
    print(ceas.dtypes)
```

```
Data Types:
sender      object
receiver    object
date        object
subject     object
body        object
label        int64
urls         int64
dtype: object
```

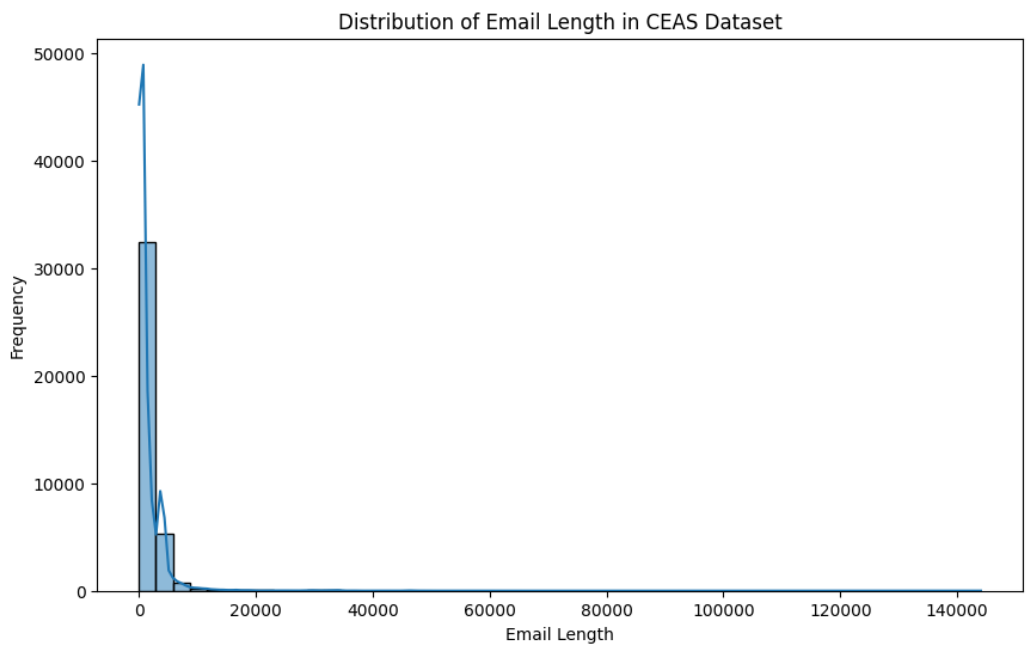
Class Distribution

```
import matplotlib.pyplot as plt
import seaborn as sns

# Class distribution
plt.figure(figsize=(6,4))
sns.countplot(x='label', data=ceas)
plt.title('Class Distribution in CEAS Dataset')
plt.xlabel('Label (0: Legitimate, 1: Phishing)')
plt.ylabel('Count')
plt.show()
```



Length of Emails

```
[ ] # Calculate length of emails
    ceas['email_length'] = ceas['subject'].str.len() + ceas['body'].str.len()

    # Plot email length distribution
    plt.figure(figsize=(10,6))
    sns.histplot(ceas['email_length'], bins=50, kde=True)
    plt.title('Distribution of Email Length in CEAS Dataset')
    plt.xlabel('Email Length')
    plt.ylabel('Frequency')
    plt.show()
```



```
[ ] # Separate word clouds for phishing and legitimate emails
    phishing_subjects = ' '.join(ceas[ceas['label'] == 1]['subject'].dropna().values)
    legit_subjects = ' '.join(ceas[ceas['label'] == 0]['subject'].dropna().values)

    # Phishing word cloud
    wordcloud_phishing = WordCloud(width=800, height=400, background_color='white').generate(phishing_subjects)
    plt.figure(figsize=(10,6))
    plt.imshow(wordcloud_phishing, interpolation='bilinear')
    plt.axis('off')
    plt.title('Common Words in Phishing Subject Lines (CEAS)')
    plt.show()

    # Legitimate word cloud
    wordcloud_legit = WordCloud(width=800, height=400, background_color='white').generate(legit_subjects)
    plt.figure(figsize=(10,6))
    plt.imshow(wordcloud_legit, interpolation='bilinear')
    plt.axis('off')
    plt.title('Common Words in Legitimate Subject Lines (CEAS)')
    plt.show()
```

{1}

Common Words in Phishing Subject Lines (CEAS)

Common Words in Legitimate Subject Lines (CEAS)

URLs and Special Characters

```
[ ] import re

    # Function to count URLs
    def count_urls(text):
        url_pattern = r'(https?://[^\s]+|http?://[^\s]+)'
        return len(re.findall(url_pattern, text))

    # Apply function to body of emails
    ceas['url_count'] = ceas['body'].apply(count_urls)

    # Plot URL count distribution
    plt.figure(figsize=(10,6))
    sns.histplot(ceas['url_count'], bins=10, kde=True)
    plt.title('Distribution of URL Count in Emails (CEAS)')
    plt.xlabel('Number of URLs')
    plt.ylabel('Frequency')
    plt.show()

    # Compare URL count between phishing and legitimate emails
    plt.figure(figsize=(10,6))
    sns.boxplot(x='label', y='url_count', data=ceas)
    plt.title('URL Count by Class (CEAS)')
    plt.xlabel('Label (0: Legitimate, 1: Phishing)')
    plt.ylabel('Number of URLs')
    plt.show()

    # Function to calculate special character ratio
    def get_special_char_ratio(text):
        special_chars = sum(not c.isalnum() and not c.isspace() for c in text)
        return special_chars / len(text) if len(text) > 0 else 0

    # Apply function to body of emails
    ceas['special_char_ratio'] = ceas['body'].apply(get_special_char_ratio)

    # Plot special character ratio distribution
    plt.figure(figsize=(10,6))
    sns.histplot(ceas['special_char_ratio'], bins=50, kde=True)
    plt.title('Distribution of Special Character Ratio in Emails (CEAS)')
    plt.xlabel('Special Character Ratio')
    plt.ylabel('Frequency')
    plt.show()

    # Compare special character ratio between phishing and legitimate emails
    plt.figure(figsize=(10,6))
    sns.boxplot(x='label', y='special_char_ratio', data=ceas)
    plt.title('Special Character Ratio by Class (CEAS)')
    plt.xlabel('Label (0: Legitimate, 1: Phishing)')
    plt.ylabel('Special Character Ratio')
    plt.show()
```

Random Forest Model

Combining and Balancing Dataset & Handling Missing Values

```
[ ]
    # Combine datasets
    datasets = [ceas, spamassassin, trec_1, trec_2, trec_3, enron, ling, nazario, nigerian_fraud]
    combined_df = pd.concat(datasets, ignore_index=True)

    # Ensure all text columns are strings
    combined_df['subject'] = combined_df['subject'].astype(str)
    combined_df['body'] = combined_df['body'].astype(str)
```

```
[ ] # Drop rows with missing values
    combined_df.dropna(inplace=True)
```

```
[ ] from sklearn.utils import resample

    # Separate majority and minority classes
    phishing = combined_df[combined_df['label'] == 1]
    legit = combined_df[combined_df['label'] == 0]

    # Balance the classes
    if len(phishing) < len(legit):
        legit_downsampled = resample(legit, replace=False, n_samples=len(phishing), random_state=42)
        balanced_df = pd.concat([phishing, legit_downsampled])
    else:
        phishing_downsampled = resample(phishing, replace=False, n_samples=len(legit), random_state=42)
        balanced_df = pd.concat([legit, phishing_downsampled])

    # Shuffle the balanced dataset
    balanced_df = balanced_df.sample(frac=1).reset_index(drop=True)
```

Features Extraction

```
[ ] # Feature extraction functions
    def extract_features(df):
        df['email_length'] = df['subject'].str.len() + df['body'].str.len()
        df['url_count'] = df['body'].apply(count_urls)
        df['special_char_ratio'] = df['body'].apply(get_special_char_ratio)
        df['urgent_action'] = df['body'].apply(check_urgent_action)
        df['poor_grammar'] = df['body'].apply(check_poor_grammar)
        df['unfamiliar_greeting'] = df['body'].apply(check_unfamiliar_greeting)
        df['requests_for_credentials'] = df['body'].apply(check_requests_for_credentials)
        df['too_good_to_be_true'] = df['body'].apply(check_too_good_to_be_true)
        df['suspicious_attachments'] = df['body'].apply(check_suspicious_attachments)
        df['html_tags'] = df['body'].apply(check_html_tags)
        df['generic_request'] = df['body'].apply(check_generic_request)
        df['polite_language'] = df['body'].apply(check_polite_language)
        df['unusual_request'] = df['body'].apply(check_unusual_request)
        df['financial_terms'] = df['body'].apply(check_financial_terms)
        df['phishing_indicators'] = df['body'].apply(check_phishing_indicators)
        df['impersonal_language'] = df['body'].apply(check_impersonal_language)
        df['repeated_words'] = df['body'].apply(check_repeated_words)
        df['abnormal_formatting'] = df['body'].apply(check_abnormal_formatting)
        df['short_and_vague'] = df['body'].apply(check_short_and_vague)
        return df

    # Apply feature extraction
    combined_df = extract_features(combined_df)

    # Convert boolean features to integers
    bool_features = ['urgent_action', 'poor_grammar', 'unfamiliar_greeting', 'requests_for_credentials', 'too_good_to_be
                     'suspicious_attachments', 'html_tags', 'generic_request', 'polite_language', 'unusual_request',
                     'financial_terms', 'phishing_indicators', 'impersonal_language', 'repeated_words', 'abnormal_format
                     'short_and_vague']

    combined_df[bool_features] = combined_df[bool_features].astype(int)

    # Selected features and labels
    features = combined_df[['email_length', 'url_count', 'special_char_ratio'] + bool_features]
    labels = combined_df['label']
```

Train-Test Split

```
[ ] from sklearn.model_selection import train_test_split

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
```

Train the Random Forest Model

```
[ ] from sklearn.ensemble import RandomForestClassifier

    # Instantiate the model
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

    # Train the model
    rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

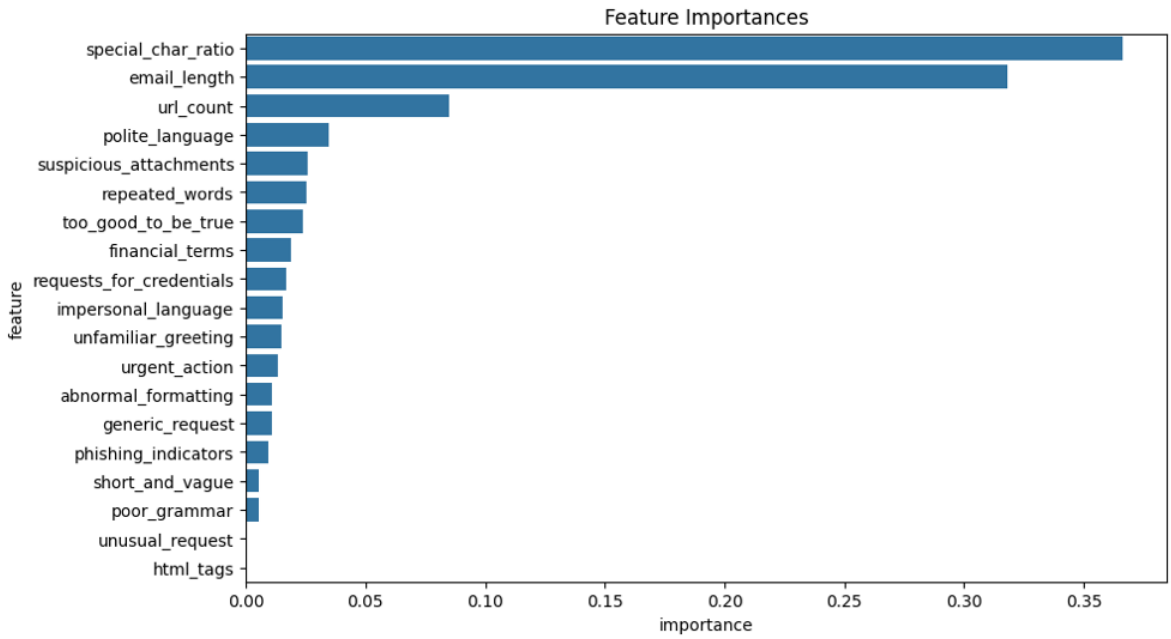
Evaluate the Model

```
[ ] from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

    # Make predictions on the test set
    y_pred = rf_model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Confusion Matrix:")
    print(conf_matrix)
```

```
Accuracy: 0.8469
Precision: 0.8523
Recall: 0.8217
F1 Score: 0.8367
Confusion Matrix:
[[18913  2826]
 [ 3539 16306]]
```

Feature Engineering

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns

    # Get feature importances
    importances = rf_model.feature_importances_
    feature_names = features.columns
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance': importances})
    feature_importances = feature_importances.sort_values(by='importance', ascending=False)

    # Plot feature importances
    plt.figure(figsize=(10,6))
    sns.barplot(x='importance', y='feature', data=feature_importances)
    plt.title('Feature Importances')
    plt.show()
```



```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Define the parameter grid for fine-tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Instantiate the RandomizedSearchCV object
rf_random = RandomizedSearchCV(estimator=rf_model,
                               param_distributions=param_grid,
                               n_iter=50,  # Number of iterations
                               cv=3,  # 3-fold cross-validation
                               verbose=2,  # Print details about progress
                               random_state=42,  # For reproducibility
                               n_jobs=-1)  # Use all available cores

# Fit the RandomizedSearchCV to the training data with selected features
rf_random.fit(X_train_selected, y_train)

# Get the best parameters
best_params = rf_random.best_params_
print(f"Best parameters found: {best_params}")

# Get the best model
best_rf_model = rf_random.best_estimator_

# Make predictions on the test set with the selected features
y_pred = best_rf_model.predict(X_test_selected)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/backend/fork_exec.py:38: RuntimeWarning: os.fork() was
  pid = os.fork()
Best parameters found: {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', '
Accuracy: 0.8456
Precision: 0.8518
Recall: 0.8156
F1 Score: 0.8333
```

Gradient Boosting Classifier

Instantiate and Train the GB Classifier Model

Import Necessary Librairies

```
[ ] from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

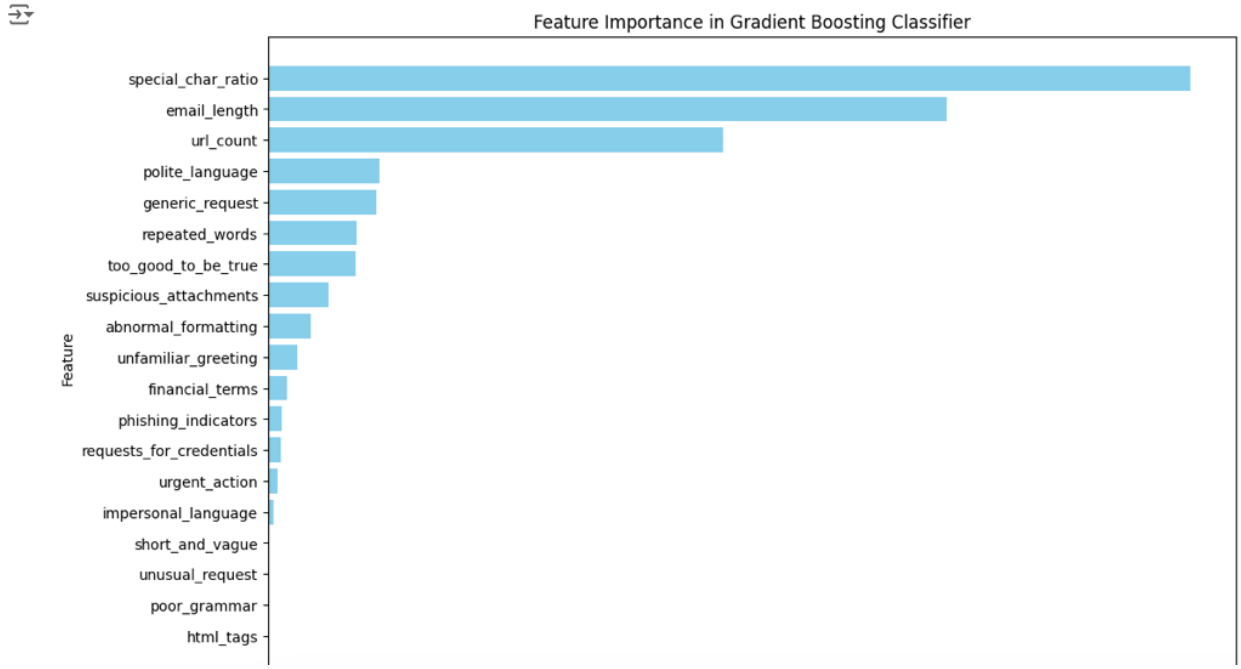
Instantiate and Train

```
[ ] # Instantiate the model
    gb_model = GradientBoostingClassifier(random_state=42)

    # Train the model
    gb_model.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

Feature Engineering

```
[ ] import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np

    # Extract feature importances
    feature_importances = gb_model.feature_importances_

    # Create a DataFrame for better visualization
    features_list = ['email_length', 'url_count', 'special_char_ratio'] + bool_features
    importance_df = pd.DataFrame({
        'Feature': features_list,
        'Importance': feature_importances
    })

    # Sort the DataFrame by importance
    importance_df = importance_df.sort_values(by='Importance', ascending=False)

    # Plot the feature importances
    plt.figure(figsize=(12, 8))
    plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.title('Feature Importance in Gradient Boosting Classifier')
    plt.gca().invert_yaxis()
    plt.show()
```



```
[ ] # Define the most important features based on the plot
    important_features = ['special_char_ratio', 'email_length', 'url_count',
                          'polite_language', 'generic_request', 'too_good_to_be_true', 'repeated_words']

    # Prepare the data with the selected features
    X_train_selected = X_train[important_features]
    X_test_selected = X_test[important_features]
```

```
[ ] # Instantiate the model
    gb_model = GradientBoostingClassifier(random_state=42)

    # Train the model
    gb_model.fit(X_train_selected, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

Evaluate the GBC Model

```
[ ] # Make predictions on the test set
    y_pred_gb = gb_model.predict(X_test_selected)

    # Calculate evaluation metrics
    accuracy_gb = accuracy_score(y_test, y_pred_gb)
    precision_gb = precision_score(y_test, y_pred_gb)
    recall_gb = recall_score(y_test, y_pred_gb)
    f1_gb = f1_score(y_test, y_pred_gb)
    conf_matrix_gb = confusion_matrix(y_test, y_pred_gb)

    print(f"Gradient Boosting Model:")
    print(f"Accuracy: {accuracy_gb:.4f}")
    print(f"Precision: {precision_gb:.4f}")
    print(f"Recall: {recall_gb:.4f}")
    print(f"F1 Score: {f1_gb:.4f}")
    print("Confusion Matrix:")
    print(conf_matrix_gb)
```

```
Gradient Boosting Model:
Accuracy: 0.7799
Precision: 0.7691
Recall: 0.7643
F1 Score: 0.7667
Confusion Matrix:
[[14140  3669]
 [ 3769 12219]]
```

Fine-Tune the GBC Model

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV

# Define the parameter grid
param_grid_gb = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Instantiate the RandomizedSearchCV object
gb_random = RandomizedSearchCV(estimator=GradientBoostingClassifier(random_state=42),
                               param_distributions=param_grid_gb,
                               n_iter=50,  # Number of iterations
                               cv=3,  # 3-fold cross-validation
                               verbose=2,  # Print details about progress
                               random_state=42,  # For reproducibility
                               n_jobs=-1)  # Use all available cores

# Fit the RandomizedSearchCV to the training data
gb_random.fit(X_train_selected, y_train)

# Get the best parameters and model
best_params_gb = gb_random.best_params_
best_gb_model = gb_random.best_estimator_

# Make predictions and evaluate the model
y_pred_gb = best_gb_model.predict(X_test_selected)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
precision_gb = precision_score(y_test, y_pred_gb)
recall_gb = recall_score(y_test, y_pred_gb)
f1_gb = f1_score(y_test, y_pred_gb)

print(f"Gradient Boosting Model - Best Parameters: {best_params_gb}")
print(f"Accuracy: {accuracy_gb:.4f}, Precision: {precision_gb:.4f}, Recall: {recall_gb:.4f}, F1 Score: {f1_gb:.4f}")
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/backend/fork_exec.py:38: RuntimeWarning: os.fork() was
  pid = os.fork()
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/backend/fork_exec.py:38: RuntimeWarning: os.fork() was
  pid = os.fork()
Gradient Boosting Model - Best Parameters: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_
Accuracy: 0.8356, Precision: 0.8394, Recall: 0.8069, F1 Score: 0.8228
```

Logistic Regression Model

Train the Logistic Regression Model

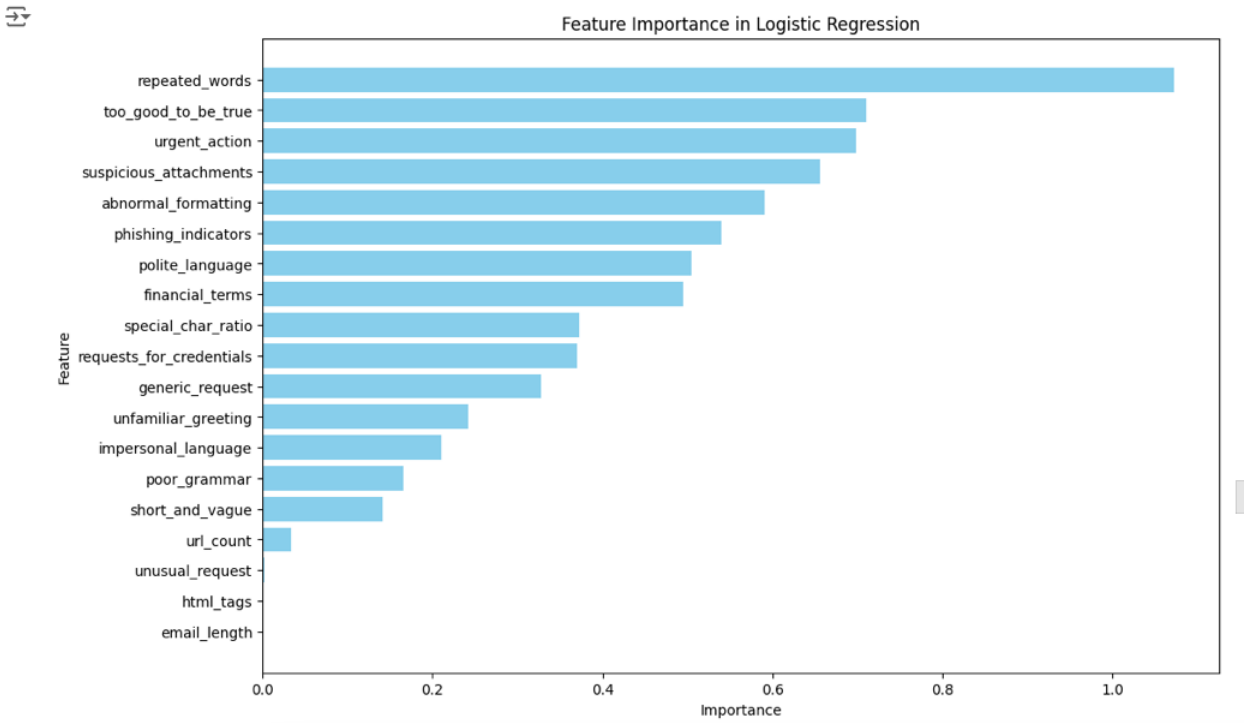
```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.linear_model import LogisticRegression
    from sklearn.preprocessing import StandardScaler

    # Standardize the features (important for Logistic Regression)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

```
[ ] from sklearn.linear_model import LogisticRegression

    # Instantiate the model
    logreg_model = LogisticRegression(random_state=42, max_iter=1000)

    # Train the model
    logreg_model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

Feature Engineering

```
[ ]
    # Extract the coefficients
    coefficients = logreg_model.coef_[0]

    # Create a DataFrame for better visualization
    feature_names =  ['email_length', 'url_count', 'special_char_ratio'] + bool_features

    importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': np.abs(coefficients)
    })

    # Sort the DataFrame by importance
    importance_df = importance_df.sort_values(by='Importance', ascending=False)

    # Plot the feature importances
    plt.figure(figsize=(12, 8))
    plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.title('Feature Importance in Logistic Regression')
    plt.gca().invert_yaxis()
    plt.show()
```



```
[ ] # Define the most important features based on the plot
    important_features = ['repeated_words', 'too_good_to_be_true', 'urgent_action', 'suspicious_attachments',
                          'abnormal_formatting', 'phishing_indicators', 'polite_language', 'financial_terms',
                          'special_char_ratio', 'requests_for_credentials']

    # Prepare the data with the selected features
    X_train_selected = X_train[important_features]
    X_test_selected = X_test[important_features]
```

```
[ ] from sklearn.linear_model import LogisticRegression

    # Instantiate the model
    logreg_model = LogisticRegression(random_state=42, max_iter=1000)

    # Train the model
    logreg_model.fit(X_train_selected, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

Evaluate the Logistic Regression Model

```
[ ] # Make predictions on the test set
    y_pred_logreg = logreg_model.predict(X_test_selected)

    # Calculate evaluation metrics
    accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
    precision_logreg = precision_score(y_test, y_pred_logreg)
    recall_logreg = recall_score(y_test, y_pred_logreg)
    f1_logreg = f1_score(y_test, y_pred_logreg)
    conf_matrix_logreg = confusion_matrix(y_test, y_pred_logreg)

    print(f"Logistic Regression Model:")
    print(f"Accuracy: {accuracy_logreg:.4f}")
    print(f"Precision: {precision_logreg:.4f}")
    print(f"Recall: {recall_logreg:.4f}")
    print(f"F1 Score: {f1_logreg:.4f}")
    print("Confusion Matrix:")
    print(conf_matrix_logreg)
```

```
Logistic Regression Model:
Accuracy: 0.6479
Precision: 0.6474
Recall: 0.5612
F1 Score: 0.6013
Confusion Matrix:
[[12923  4886]
 [ 7015  8973]]
```

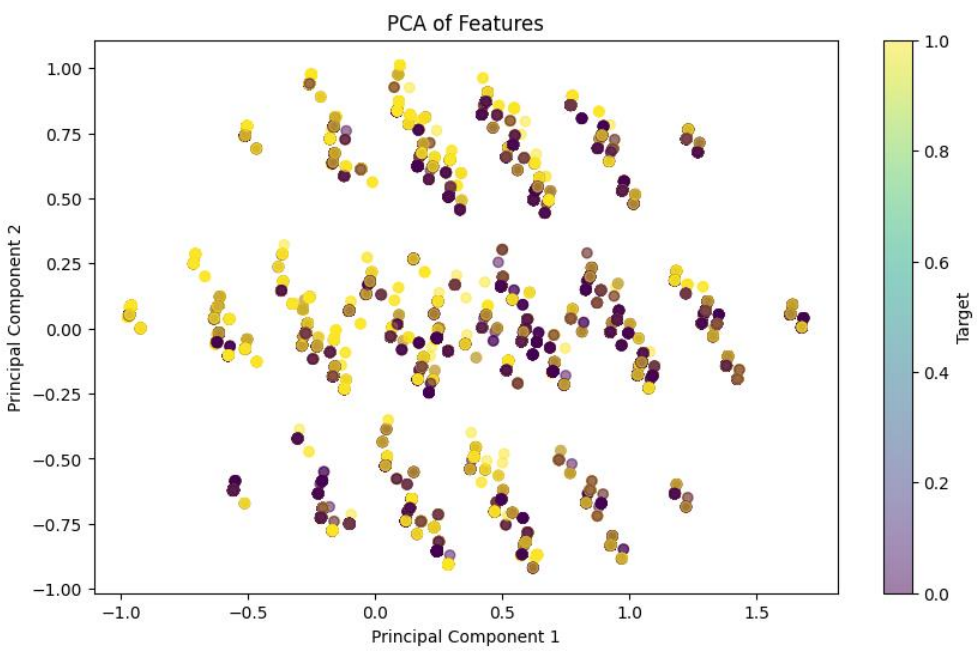
Fine-Tune the Logistic Regression Model

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import RandomizedSearchCV

    # Define the parameter grid
    param_grid_logreg = {
        'C': [0.01, 0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear', 'saga']
    }

    # Instantiate the RandomizedSearchCV object
    logreg_random = RandomizedSearchCV(estimator=LogisticRegression(random_state=42),
                                       param_distributions=param_grid_logreg,
                                       n_iter=50,  # Number of iterations
                                       cv=3,  # 3-fold cross-validation
                                       verbose=2,  # Print details about progress
                                       random_state=42,  # For reproducibility
                                       n_jobs=-1)  # Use all available cores

    # Fit the RandomizedSearchCV to the training data
    logreg_random.fit(X_train_selected, y_train)

    # Get the best parameters and model
    best_params_logreg = logreg_random.best_params_
    best_logreg_model = logreg_random.best_estimator_

    # Make predictions and evaluate the model
    y_pred_logreg = best_logreg_model.predict(X_test_selected)
    accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
    precision_logreg = precision_score(y_test, y_pred_logreg)
    recall_logreg = recall_score(y_test, y_pred_logreg)
    f1_logreg = f1_score(y_test, y_pred_logreg)

    print(f"Logistic Regression Model - Best Parameters: {best_params_logreg}")
    print(f"Accuracy: {accuracy_logreg:.4f}, Precision: {precision_logreg:.4f}, Recall: {recall_logreg:.4f}, F1 Score: {
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:305: UserWarning: The total space of param
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/backend/fork_exec.py:38: RuntimeWarning: os.fork() was
  pid = os.fork()
Logistic Regression Model - Best Parameters: {'solver': 'liblinear', 'penalty': 'l2', 'C': 100}
Accuracy: 0.6485, Precision: 0.6479, Recall: 0.5630, F1 Score: 0.6025
```

Explanation of Performance

```
[ ] from sklearn.decomposition import PCA

    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_train_selected)

    plt.figure(figsize=(10, 6))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, cmap='viridis', alpha=0.5)
    plt.title('PCA of Features')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.colorbar(label='Target')
    plt.show()

    #Homogeneous data appear clustered together with no clear separation between classes.
```



DistilBERT Model

Feature Engineering

[ ] ↳ 6 cells hidden

Combining and Balancing the Datasets

[ ] ↳ 4 cells hidden

Create the Dataset Class

```
[ ] class EmailDataset(Dataset):
        def __init__(self, texts, labels, additional_features, tokenizer, max_len):
            self.texts = texts.reset_index(drop=True)
            self.labels = labels.reset_index(drop=True)
            self.additional_features = additional_features.reset_index(drop=True)
            self.tokenizer = tokenizer
            self.max_len = max_len

        def __len__(self):
            return len(self.texts)
        def __getitem__(self, item):
            text = str(self.texts[item])
            label = self.labels[item]
            additional_features = self.additional_features.iloc[item].values.astype(float)
            encoding = self.tokenizer.encode_plus(
                text,
                add_special_tokens=True,
                max_length=self.max_len,
                return_token_type_ids=False,
                padding='max_length',
                truncation=True,
                return_attention_mask=True,
                return_tensors='pt',
            )
            return {
                'text': text,
                'input_ids': encoding['input_ids'].flatten(),
                'attention_mask': encoding['attention_mask'].flatten(),
                'additional_features': torch.tensor(additional_features, dtype=torch.float),
                'label': torch.tensor(label, dtype=torch.long)
            }
```

Split the Data into Train and Test Sets

```
[ ] feature_columns = [
        'urgent_action', 'poor_grammar', 'unfamiliar_greeting', 'requests_for_credentials', 'too_good_to_be_true',
        'suspicious_attachments', 'html_tags', 'length', 'special_char_ratio', 'url_count', 'generic_request',
        'polite_language', 'unusual_request', 'financial_terms', 'phishing_indicators', 'impersonal_language',
        'repeated_words', 'abnormal_formatting', 'short_and_vague'
    ]

    train_texts, test_texts, train_labels, test_labels, train_additional_features, test_additional_features = train_test
        balanced_df['text'], balanced_df['label'], balanced_df[feature_columns], test_size=0.1, random_state=42
    )

    # Train and test datasets
    tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-multilingual-cased')

    class EmailDataset(Dataset):
        def __init__(self, texts, labels, additional_features, tokenizer, max_len):
            self.texts = texts.reset_index(drop=True)
            self.labels = labels.reset_index(drop=True)
            self.additional_features = additional_features.reset_index(drop=True)
            self.tokenizer = tokenizer
            self.max_len = max_len

        def __len__(self):
            return len(self.texts)

        def __getitem__(self, item):
            text = str(self.texts[item])
            label = self.labels[item]
            additional_features = self.additional_features.iloc[item].values.astype(float)
            encoding = self.tokenizer.encode_plus(
                text,
                add_special_tokens=True,
                max_length=self.max_len,
                return_token_type_ids=False,
                padding='max_length',
                truncation=True,
                return_attention_mask=True,
                return_tensors='pt',
            )
            return {
                'text': text,
                'input_ids': encoding['input_ids'].flatten(),
                'attention_mask': encoding['attention_mask'].flatten(),
                'additional_features': torch.tensor(additional_features, dtype=torch.float),
                'label': torch.tensor(label, dtype=torch.long)
            }

    train_dataset = EmailDataset(train_texts, train_labels, train_additional_features, tokenizer, max_len=128)
    test_dataset = EmailDataset(test_texts, test_labels, test_additional_features, tokenizer, max_len=128)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132: FutureWarning: `resume_download` is de
  warnings.warn(
```

Evaluate the Model

```
[ ] def compute_metrics(p):
        pred, labels = p
        pred = np.argmax(pred, axis=1)
        accuracy = accuracy_score(labels, pred)
        precision, recall, f1, _ = precision_recall_fscore_support(labels, pred, average='binary')
        return {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
        }

    trainer = CustomTrainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset,
        compute_metrics=compute_metrics
    )

    evaluation_results = trainer.evaluate()
    print(evaluation_results)
```

```
[992/992 02:07]
{'eval_loss': 0.10671290755271912, 'eval_accuracy': 0.9808371154815936, 'eval_precision': 0.9805408137477887, 'eval_r
```

```python
# Initialize the Flask app
app = Flask(__name__)
CORS(app)  # Enable CORS for the entire app

# Define the custom model class
class CustomDistilBertForSequenceClassification(nn.Module):
    def __init__(self, num_labels, additional_features_dim):
        super(CustomDistilBertForSequenceClassification, self).__init__()
        self.bert = DistilBertModel.from_pretrained('distilbert-base-multilingual-cased')
        self.dropout = nn.Dropout(0.3)
        self.classifier = nn.Linear(self.bert.config.hidden_size + additional_features_dim, num_labels)

    def forward(self, input_ids, attention_mask, additional_features, labels=None):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs.last_hidden_state[:, 0]  # Use the first token's representation
        pooled_output = self.dropout(pooled_output)
        combined_output = torch.cat((pooled_output, additional_features), dim=1)
        logits = self.classifier(combined_output)

        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, self.classifier.out_features), labels.view(-1))

        return {'logits': logits, 'loss': loss}

# Load the model and tokenizer
model_path = './phishing_detector_v2'
tokenizer = DistilBertTokenizer.from_pretrained(model_path)

# Load the model configuration
model_config_path = f'{model_path}/model_config.bin'
model_config = torch.load(model_config_path)

# Instantiate the custom model
model = CustomDistilBertForSequenceClassification(
    num_labels=model_config['num_labels'],
    additional_features_dim=model_config['additional_features_dim']
)

# Load the state dictionary
model_state_path = f'{model_path}/pytorch_model.bin'
model.load_state_dict(torch.load(model_state_path, map_location=torch.device('cpu')))
model.eval()  # Set the model to evaluation mode

# Define the feature extraction functions
def check_urgent_action(text):
    urgent_keywords = [
        'urgent', 'immediately', 'asap', 'action required', 'important', 'critical', 'attention', 'prompt',
        'emergency', 'act now', 'time-sensitive', 'alert', 'high-priority', 'important update', 'do not ignore',
        'response needed', 'please respond'
    ]
    return any(keyword in text.lower() for keyword in urgent_keywords)

def check_poor_grammar(text):
    spelling_errors = [
        'teh', 'recieve', 'adn', 'definately', 'seperate', 'occured', 'untill', 'adress', 'wich', 'freind',
        'interrest', 'recieved', 'tommorow', 'hapen', 'acomodate', 'comitment', 'gratefull', 'inconvience',
        'posession', 'pronounciation', 'publically', 'recieve', 'untill', 'wierd', 'occurred', 'adres',
        'adressee', 'adress', 'suposed', 'supposse', 'succesfull', 'accomodate', 'acomodate', 'acommadate'
    ]
    return any(error in text.lower() for error in spelling_errors)

def check_unfamiliar_greeting(text):
    greetings = [
        'dear customer', 'dear user', 'hello', 'hi', 'dear sir', 'dear madam', 'to whom it may concern',
        'dear valued customer', 'greetings', 'hey', 'good day', 'hi there', 'dear friend'
    ]
    return any(greeting in text.lower() for greeting in greetings)
```

```python
def check_unfamiliar_greeting(text):
    greetings = [
        'dear customer', 'dear user', 'hello', 'hi', 'dear sir', 'dear madam', 'to whom it may concern',
        'dear valued customer', 'greetings', 'hey', 'good day', 'hi there', 'dear friend'
    ]
    return any(greeting in text.lower() for greeting in greetings)

def check_requests_for_credentials(text):
    credentials_requests = [
        'login', 'password', 'ssn', 'social security', 'credit card', 'payment information',
        'username', 'credentials', 'account', 'access', 'verify', 'update', 'confirm',
        'authentication', 'secure', 'personal information', 'pin', 'two-factor authentication'
    ]
    return any(request in text.lower() for request in credentials_requests)

def check_too_good_to_be_true(text):
    too_good_to_be_true = [
        'won', 'winner', 'prize', 'free', 'gift', 'congratulations', 'jackpot', 'fortune', 'wealth',
        'cash', 'bonanza', 'reward', 'gain', 'profitable', 'lucrative', 'exclusive offer',
        'limited time', (parameter) text: Any                    r', 'no catch', 'guaranteed', 'win big',
        'golden opportun                                         vestment opportunity'
    ]
    return any(offer in text.lower() for offer in too_good_to_be_true)

def check_suspicious_attachments(text):
    suspicious_attachments = [
        'attachment', 'file', 'download', 'click', 'open', 'see attached', 'document', 'invoice',
        'statement', 'receipt', 'urgent', 'important', 'security', 'alert', 'click here',
        'check attachment', 'attached file', 'view document', 'access file'
    ]
    return any(attachment in text.lower() for attachment in suspicious_attachments)

def check_html_tags(text):
    html_tags = ['<a', '<img', '<form', '<input', '<script', '<iframe']
    return any(tag in text.lower() for tag in html_tags)

def get_length(text):
    return len(text)

def get_special_char_ratio(text):
    special_chars = sum(not c.isalnum() and not c.isspace() for c in text)
    return special_chars / len(text)

def count_urls(text):
    url_pattern = r'(https?://[^\s]+|http?://[^\s]+)'
    urls = re.findall(url_pattern, text)
    return len(urls)

def check_generic_request(text):
    generic_phrases = [
        'as discussed', 'could you forward', 'please send', 'requesting document', 'quick info', 'need info',
        'can you send', 'provide details', 'send the file', 'urgent request', 'follow up', 'needed asap',
        'send me the document'
    ]
    return any(phrase in text.lower() for phrase in generic_phrases)

def check_polite_language(text):
    polite_phrases = ['please', 'kindly', 'could you', 'thank you', 'would you', 'grateful', 'appreciate', 'regards', 'sincerely']
    return any(phrase in text.lower() for phrase in polite_phrases)

def check_unusual_request(text):
    unusual_requests = ['gift cards', 'transfer money', 'bank details', 'send funds', 'urgent transfer', 'wire money', 'cryptocurrency']
    return any(request in text.lower() for request in unusual_requests)

def check_financial_terms(text):
    financial_terms = ['invoice', 'payment', 'credit', 'debit', 'bank', 'account', 'fund', 'transaction', 'balance', 'statement', 'due']
    return any(term in text.lower() for term in financial_terms)

def check_phishing_indicators(text):
    phishing_indicators = ['phishing', 'scam', 'fraud', 'fake', 'hoax', 'spoof', 'malware', 'virus', 'identity theft']
    return any(indicator in text.lower() for indicator in phishing_indicators)
```

```python
def check_impersonal_language(text):
    impersonal_phrases = ['user', 'account holder', 'valued customer', 'member', 'client', 'account owner', 'subscriber']
    return any(phrase in text.lower() for phrase in impersonal_phrases)

def check_repeated_words(text):
    words = text.lower().split()
    return any(words.count(word) > 2 for word in words)

def check_abnormal_formatting(text):
    formatting_issues = ['ALL CAPS', '!!!', '???', 'red text', 'bold', 'italic']
    return any(issue in text for issue in formatting_issues)

def check_short_and_vague(text):
    return len(text) < 50 or 'important' in text.lower()

# Preprocess input
def preprocess_input(email_content):
    features = {
        'urgent_action': check_urgent_action(email_content),
        'poor_grammar': check_poor_grammar(email_content),
        'unfamiliar_greeting': check_unfamiliar_greeting(email_content),
        'requests_for_credentials': check_requests_for_credentials(email_content),
        'too_good_to_be_true': check_too_good_to_be_true(email_content),
        'suspicious_attachments': check_suspicious_attachments(email_content),
        'html_tags': check_html_tags(email_content),
        'length': get_length(email_content),
        'special_char_ratio': get_special_char_ratio(email_content),
        'url_count': count_urls(email_content),
        'generic_request': check_generic_request(email_content),
        'polite_language': check_polite_language(email_content),
        'unusual_request': check_unusual_request(email_content),
        'financial_terms': check_financial_terms(email_content),
        'phishing_indicators': check_phishing_indicators(email_content),
        'impersonal_language': check_impersonal_language(email_content),
        'repeated_words': check_repeated_words(email_content),
        'abnormal_formatting': check_abnormal_formatting(email_content),
        'short_and_vague': check_short_and_vague(email_content),
    }
    encoding = tokenizer.encode_plus(
        email_content,
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors='pt',
    )
    return encoding, features

# Predict function
def predict(email_content):
    encoding, features = preprocess_input(email_content)
    input_ids = encoding['input_ids']
    attention_mask = encoding['attention_mask']
    additional_features = torch.tensor(list(features.values()), dtype=torch.float).unsqueeze(0)
    outputs = model(input_ids=input_ids, attention_mask=attention_mask, additional_features=additional_features)
    logits = outputs['logits']
    probabilities = F.softmax(logits, dim=1)
    phishing_probability = probabilities[0][1].item()
    return phishing_probability
```

```python
# Predict function
def predict(email_content):
    encoding, features = preprocess_input(email_content)
    input_ids = encoding['input_ids']
    attention_mask = encoding['attention_mask']
    additional_features = torch.tensor(list(features.values()), dtype=torch.float).unsqueeze(0)
    outputs = model(input_ids=input_ids, attention_mask=attention_mask, additional_features=additional_features)
    logits = outputs['logits']
    probabilities = F.softmax(logits, dim=1)
    phishing_probability = probabilities[0][1].item()
    return phishing_probability

@app.route('/detect', methods=['POST'])
def detect_phishing():
    data = request.get_json()
    email_content = data.get('content', '')

    if not email_content:
        return jsonify({'result': 'Error: No content provided.'}), 400

    phishing_probability = predict(email_content)
    return jsonify({'phishing_probability': phishing_probability})

if __name__ == '__main__':
    app.run(debug=True)
```