AI-Powered Phishing Email Detection System

COS720 - Computer Information and Security I

Kumbirai Shonhiwa
University of Pretoria

May 2025

Contents

1 Introduction

Phishing detection utilising Artificial Intelligence (AI) involves the use of Machine Learning (ML) and Deep Learning (DL) algorithms to automatically identify and categorise fraudulent emails. These AI techniques analyse email content to detect threats, playing a vital role in mitigating email-related cyberattacks by identifying harmful messages that exploit human trust.[4] The following is a summary of how AI is applied to phishing detection and its advantages over conventional approaches:

- Machine learning and deep learning are used to automatically detect phishing emails, thereby improving email security. [4]
- Machine learning methods classify emails using features like sender details, content structure, and metadata. [4]
- DL frameworks, including CNNs, RNNs, LSTMs, and notably transformer-based models like BERT and RoBERTa, automatically extract complex features from raw email content. These DL models evaluate the contextual and semantic nuances of emails, resulting in improved accuracy and robustness. [4]
- AI-based solutions can effectively detect advanced phishing techniques, including those created with AI tools. [4]
- Traditional detection methods that rely primarily on static features, such as URLs or IP addresses, often struggle to identify AI-generated phishing emails that closely imitate legitimate communication.
- Deep learning frameworks consistently outperform traditional machine learning methods in terms of detection accuracy and adaptability. [4]
- Transformer models like BERT and RoBERTa have achieved accuracies of 98.99% and 99.08% on a balanced dataset. [4]
- These DL frameworks outperformed conventional ML methods by an average margin of 4.7%. [4]
- AI-driven solutions are considered essential for strengthening modern email security systems. [4]

2 Dataset Preparation

This study utilises multiple phishing email datasets from Kaggle [1]. The datasets were preprocessed to extract relevant features for phishing detection, including:

- Subject line analysis (keywords, patterns, length)
- Sender information (domain reputation, address structure)
- Email body content (suspicious links, HTML elements, text-to-link ratio)
- Structural metadata (headers, attachments, encoding)

2.1 Dataset Exploration

We analysed seven distinct email datasets with varying characteristics, as shown in Table 1.

Table 1: Composition of Email Datasets

Dataset	Phishing Emails	Legitimate Emails	Total
SpamAssassin	1,718	4,091	5,809
Nigerian Fraud	3,332	0	3,332
Phishing Email	42,891	39,595	82,486
CEAS_08	21,842	17,312	39,154
Enron	13,976	15,791	29,767
Ling	458	2,401	2,859
Nazario	1,565	0	1,565
Total	**85,782**	**79,190**	**164,972**

2.2 Data Preprocessing

Several preprocessing steps were applied to standardise the input data:

1. Text normalisation : Converting to lowercase, removing special characters, and stemming
2. Feature extraction: Identifying URLs, analysing email structure, and extracting header information
3. Balancing: Addressing class imbalance through stratified sampling techniques
4. Dimensionality reduction: Selecting the most relevant features based on statistical significance

2.3 Feature Engineering

For effective phishing detection, we combined multiple text-based features from each email into a unified representation. This approach allows the model to identify patterns across different email components simultaneously.

The feature engineering process concatenates four key components of each email:

- Sender information: Email addresses and display names that may contain phishing indicators
- Subject line: Often contains social engineering cues or urgent language
- Email body: The main content where phishing attempts are elaborated
- URLs: Web links that might redirect to malicious websites

By combining these features into a single text representation (`X_combined`), we enable the model to capture interactions between different email components. For example, identifying when suspicious URLs complement social engineering narratives in the body text. The prepared datasets were then split into training (70%), validation (15%), and testing (15%) sets while maintaining the distribution of phishing and legitimate emails.

3 Model Development

This section outlines our process for developing a machine learning model to classify phishing emails. We start by selecting the appropriate algorithms, then compare each alternative algorithm with our chosen one, and finally, we describe our evaluation methodology.

Process of Selecting a Model for Phishing Email Classification

Selecting the right machine learning model is essential for developing an effective phishing email classification system. We evaluated several supervised learning classifiers, including Support Vector Machines (SVM), Naive Bayes, Decision Trees, and Neural Networks.

Comparing SVMs with Other Classifiers

Support Vector Machines excel in handling high-dimensional and complex data, such as email content. Unlike Naive Bayes, which assumes feature independence and struggles with non-linearity, SVMs can model these relationships using kernel functions.[9, 11] However, they require careful hyperparameter tuning and can be more computationally intensive. Compared to Decision Trees, SVMs better manage high-dimensional data and are less likely to overfit. Decision Trees are simpler but can become unstable with noisy datasets.[9, 10, 11] SVMs are also more resistant to overfitting than Neural Networks, which offer greater flexibility but require larger datasets and more computational resources.[8, 9, 13]

Evaluating and Selecting an Algorithm

We used phishing and legitimate emails from the CEAS 2008 corpus for our dataset, applying TF-IDF vectorisation to extract features like the sender's email address, subject line, body, and URLs. We evaluated model performance through stratified k-fold cross-validation, utilising metrics such as accuracy, precision, recall, and F1-score. The Support Vector Machine (SVM) provided the best precision-recall balance, minimising false negatives in phishing detection. We optimised performance by fine-tuning hyperparameters through grid and randomised searches.

3.1 Model Training Approach

Our model is trained using supervised learning with labelled data from multiple phishing email datasets. The training process involves:

1. Feature Extraction: Converting raw email content into numerical features using TF-IDF vectorisation, focusing on sender information, subject lines, email body, and embedded URLs.
2. Data Preprocessing: Handling missing values, removing stop words, and normalising text data to improve feature quality.
3. Class Imbalance Handling: Applying Random Undersampling to the majority class to create a more balanced training dataset.
4. Hyperparameter Tuning: Using techniques such as grid search or random search to optimise model hyperparameters for improved performance.

3.2 Performance Evaluation

The performance of our models is evaluated using a comprehensive set of metrics to account for the classification task's nuances and the potential imbalance in the dataset:

Table 2: Classification Report (Accuracy = 0.99)

Class	**Precision**	**Recall**	**F1-score**	**Support**
0 (Legitimate)	0.9910	0.9873	0.9891	3,462
1 (Phishing)	0.9900	0.9929	0.9914	4,369
Macro avg	0.9905	0.9901	0.9903	7,831
Weighted avg	0.9904	0.9904	0.9904	7,831

Table 3 summarises the classification performance of the model in detecting legitimate versus phishing emails. The metrics shown are:

- **Precision:** Precision for each class measures the proportion of correctly predicted samples among all samples predicted as that class. For example, a precision of 0.99 for the legitimate class means that 99.1% of predicted legitimate emails were truly legitimate.
- **Recall:** The proportion of actual samples of each class that were correctly identified by the model. A recall of 0.9929 for the phishing class means 99.29% of phishing emails were correctly detected.
- **F1-score:** The harmonic mean of precision and recall, providing a balanced measure of model performance.
- **Support:** The total number of true samples for each class in the test set.

The model achieves an accuracy of 99%, correctly classifying 99% of emails. Both classes show high precision, recall, and F1-scores, indicating effective identification of phishing emails with few false positives and negatives. The macro average is the unweighted mean of metrics, while the weighted average considers class distribution.

Confusion Matrix: The model achieved high classification performance, as shown by the confusion matrix with 4338 true positives and 3418 true negatives, and very few false positives (44) and false negatives (31). The precision-recall curve remains close to (1.0, 1.0), indicating that the model maintains excellent precision and recall across thresholds. [Figure 1]

Precision-Recall Curve: The curve demonstrates that the model sustains a high precision level even as recall increases, highlighting its ability to detect most spam emails with minimal false alarms. The slight drop near full recall is expected due to the precision-recall trade-off. [Figure 2]

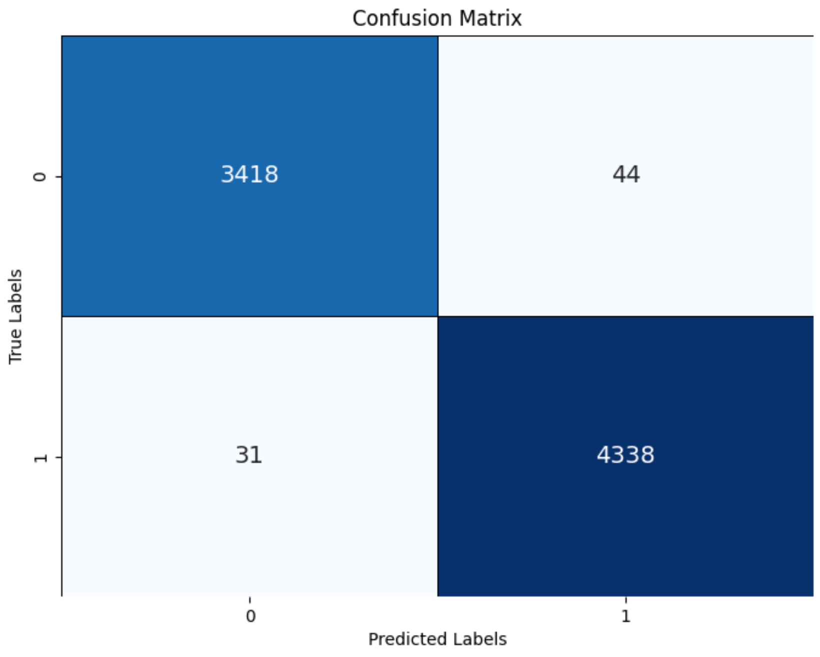


Figure 1: Confusion Matrix

3.3 Feature Interpretation

For the final selected model, we implement a feature interpretation mechanism to understand which email characteristics most strongly influence the classification decision:

```
# Interpretation: extract non-zero TF-IDF features
feature_index = X_email.nonzero()[1]
tfidf_scores = X_email.data
word_importance = [(feature_names[i], coef[i] * tfidf_scores[j])
                   for j, i in enumerate(feature_index)]

word_importance_sorted = sorted(word_importance, key=lambda x: abs(x[1]), reverse=Tru
```

This interpretation capability enhances the practical utility of the model by providing insights into the decision-making process and highlighting potentially suspicious elements in emails flagged as phishing attempts.

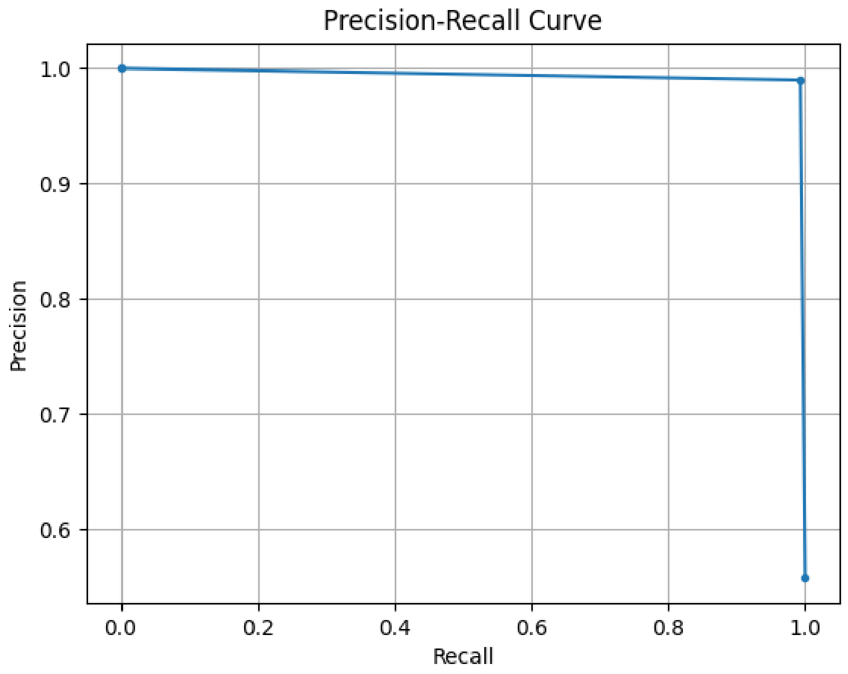


Figure 2: Precision-Recall Curve

4 Prototype Implementation

This section details the development of a web-based prototype interface for the phishing email detection system. The implementation provides an accessible platform for users to interact with the trained machine learning model through an intuitive user interface.

4.1 Functional Requirements

The phishing email classification system shall fulfil the following core functional requirements:

- **Data Ingestion:** The system shall load and process multiple datasets, including CEAS_08, Nigerian_Fraud, Enron, Ling, and others, with support for CSV file formats.
- **Preprocessing:** The system shall extract textual features (e.g., `sender`, `subject`, `body`, `urls`) and apply TF-IDF vectorisation to convert text data into numerical form suitable for machine learning.
- **Model Training:** The system shall train a Support Vector Machine (SVM) classifier using a linear kernel. It shall support undersampling of the majority class to handle imbalanced datasets.
- **Evaluation:** The system shall evaluate the model using accuracy, precision, recall, F1-score, confusion matrix, and a precision-recall curve. Results will be displayed both numerically and visually.

- **Model Persistence:** The trained model shall be saved using `joblib` for later deployment or testing.
- **Prediction and Interpretation:** The system shall allow batch classification of new email samples and provide interpretation of predictions by highlighting top contributing features based on TF-IDF and SVM coefficients.
- **Web Integration:** A lightweight web interface using Flask shall be implemented to allow user interaction for real-time email classification and display of prediction results.[14]

4.2 System Architecture

The prototype implementation follows a standard client-server architecture, with the following components:

- Back-end Server: Implemented using Flask, a lightweight Python web framework that provides the necessary tools for routing HTTP requests, handling form submissions, and serving HTML templates. [14]
- Machine Learning Model: The trained SVM model is integrated into the back-end server, along with the TF-IDF vectorizer used for feature extraction.[13]
- Web Interface: A responsive HTML/CSS frontend that allows users to input email text or upload email files for classification.

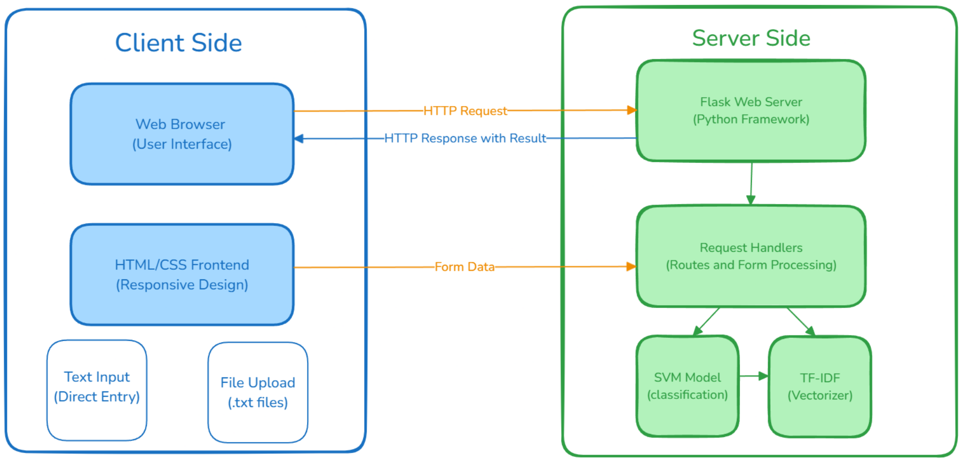

Client Side
Web Browser (User Interface)
HTML/CSS Frontend (Responsive Design)
Text Input (Direct Entry)
File Upload (.txt files)
HTTP Request
HTTP Response with Result
Form Data
Server Side
Flask Web Server (Python Framework)
Request Handlers (Routes and Form Processing)
SVM Model (classification)
TF-IDF (Vectorizer)

Figure 3: Client-Sever Architecture Diagram

4.3 Flask Application Development

The core of the prototype is a Flask application that handles user requests, processes email content, and returns classification results. The application is developed using Python, leveraging the Flask framework for web server functionality and various scientific libraries for machine learning tasks.[13]

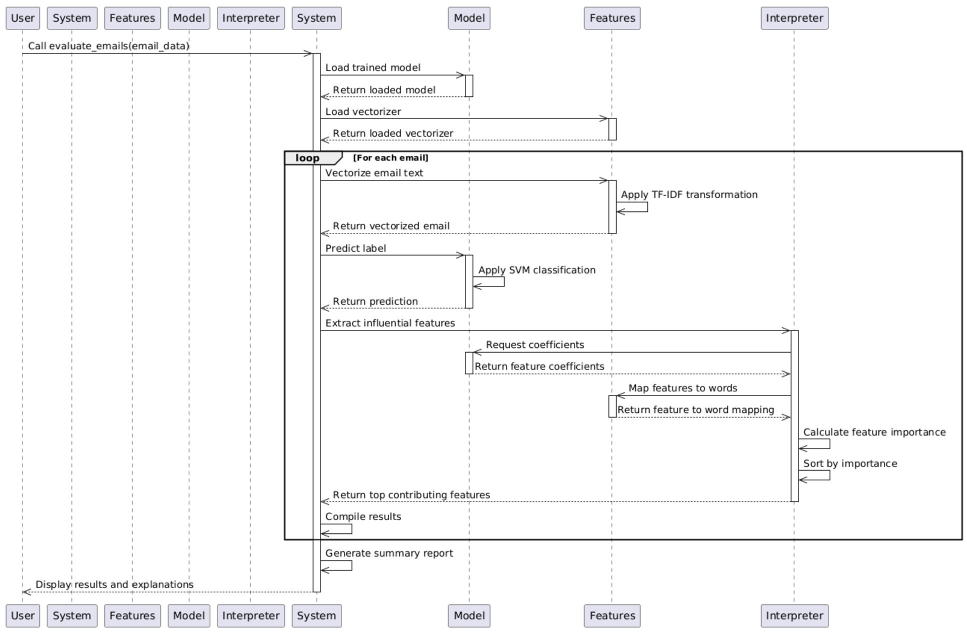


Figure 4: Sequence Diagram of Email Evaluation

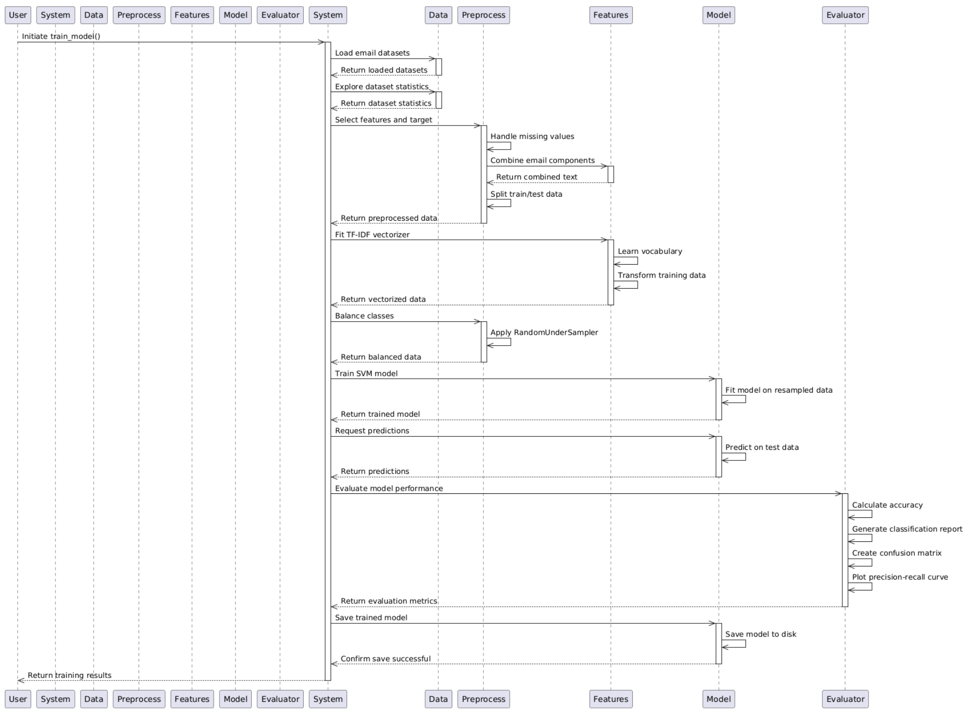


Figure 5: Sequence Diagram of Model Training Process

4.3.1 Application Initialization

The initialisation process includes:

- Loading the necessary Python libraries, including Flask for web server functionality, joblib for model loading, scikit-learn for text vectorisation, and pandas for data handling.
- Loading the pre-trained SVM model from a serialised file (`svm_model.pkl`), which

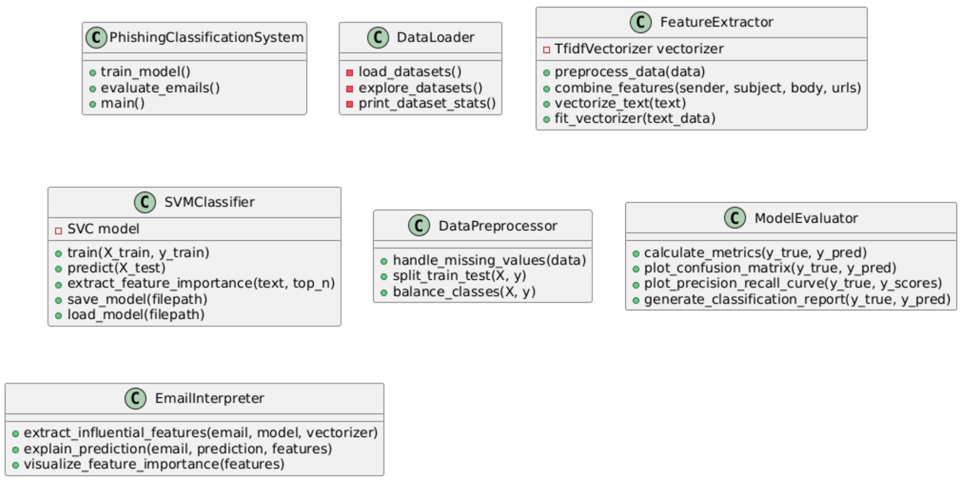

PhishingClassificationSystem
train_model()
evaluate_emails()
main()
DataLoader
load_datasets()
explore_datasets()
print_dataset_stats()
FeatureExtractor
TfidfVectorizer vectorizer
preprocess_data(data)
combine_features(sender, subject, body, urls)
vectorize_text(text)
fit_vectorizer(text_data)
SVMClassifier
SVC model
train(X_train, y_train)
predict(X_test)
extract_feature_importance(text, top_n)
save_model(filepath)
load_model(filepath)
DataPreprocessor
handle_missing_values(data)
split_train_test(X, y)
balance_classes(X, y)
ModelEvaluator
calculate_metrics(y_true, y_pred)
plot_confusion_matrix(y_true, y_pred)
plot_precision_recall_curve(y_true, y_scores)
generate_classification_report(y_true, y_pred)
EmailInterpreter
extract_influential_features(email, model, vectorizer)
explain_prediction(email, prediction, features)
visualize_feature_importance(features)

Figure 6: Class Diagram of Model

contains the trained model parameters from the training phase.

- Initialising and fitting the TF-IDF vectoriser with the same parameters used during model training to ensure consistent feature extraction.
- Creating a new Flask application instance, which will handle HTTP requests and serve the web interface.

4.3.2 Email Evaluation Function

The core functionality for email classification is encapsulated in the evaluate emails function, which performs several key operations:

- Accepts email data in either dictionary or DataFrame format, with fields for email descriptors, text content, and true labels (though the latter is only used for logging in the prototype).
- Converts the input email text to TF-IDF feature vectors using the pre-trained vectorizer, ensuring the same feature extraction process used during model training.
- Applies the trained SVM model to predict whether each email is phishing (1) or legitimate (0).
- Logs detailed results for debugging and monitoring purposes.
- Returns a human-readable classification result ('Phishing' or 'Non-Phishing') for the user interface.

4.3.3 Web Routes and Request Handling

The Flask application defines two main routes to handle user interactions:

- A root route (/) that serves the main page of the application, rendering the `index.html` template.

- A prediction route (`/predict`) that accepts POST requests containing email content for classification.
- Dual input methods, allowing users to either type/paste email text directly into a form field or upload a text file containing the email.
- Input validation to ensure that valid email content is provided before attempting classification.
- Integration with the evaluation function to classify the provided email content.
- Rendering of the results page, which includes the classification outcome.

4.4 Web Interface Design

The web interface is designed to be user-friendly and intuitive, focusing on simplicity and clear presentation of results. The interface is implemented using HTML and CSS, with the following key components:

- Input Form: A form that allows users to input email content either by typing/pasting text or by uploading a text file.
- Submission Button: A clearly labelled button to submit the email for classification.
- Results Display: A section that shows the classification result (phishing or legitimate) along with a confidence score.
- Error Messages: Clear feedback when inputs are invalid or processing fails.

Figure 7 shows a conceptual mock-up of the web interface design.

4.5 Application Deployment

The application is run using the following code at the end of the Flask script:

Listing 1: Application Deployment

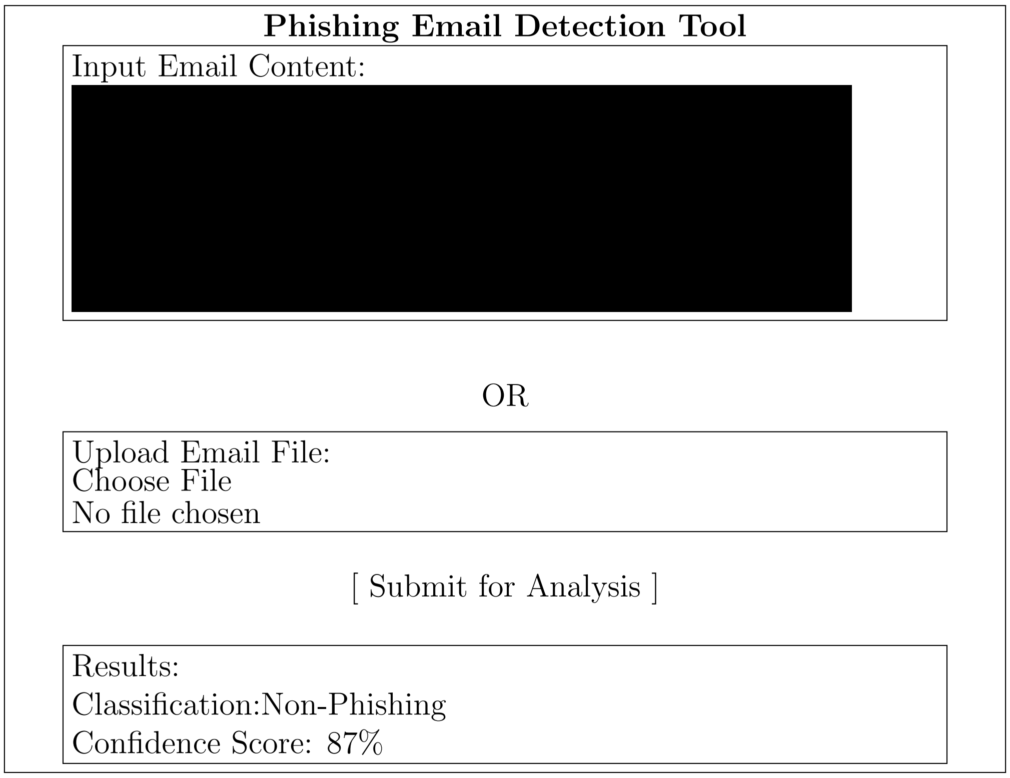
```
if name == 'main':
app.run(debug=True)
```

For development purposes, the application is run with debug mode enabled, which provides detailed error messages and automatic server reloading when code changes are detected.

4.6 System Features and Capabilities

The prototype implementation offers several key features and capabilities designed with a focus on usability, robustness, and security:

- **Email Classification**: The core functionality involves classifying emails as either phishing or legitimate using a trained Support Vector Machine (SVM) model in conjunction with TF-IDF vectorisation. This enables accurate content-based analysis.



Phishing Email Detection Tool

Input Email Content:

OR

Upload Email File:
Choose File
No file chosen

[Submit for Analysis]

Results:
Classification:Non-Phishing
Confidence Score: 87%

Figure 7: Conceptual design of the web interface for phishing email detection

- **Dual Input Methods**: Users can submit email content either through direct text input or by uploading a plain text file (.txt). This flexibility accommodates varied user scenarios while ensuring safe file handling.
- **Result Visualisation**: Classification results are presented clearly, with intuitive visual indicators (e.g., colour-coded labels) to distinguish phishing from legitimate emails.
- **Input Validation and Sanitisation**: All user input is thoroughly validated and sanitised to guard against malicious content. This includes checking for harmful patterns like embedded scripts, limiting the length of input, and ensuring that files match expected formats.
- **Security Measures**: The system implements various security measures, including rate limiting to prevent abuse, secure session management (such as HTTP-only and secure cookies), restrictions on file types and sizes, and safeguards against path traversal in file uploads.
- **Logging and Monitoring**: All system activities, including errors, input anomalies, and user interactions, are logged using a secure rotating log mechanism. This process facilitates debugging, security auditing, and system monitoring without revealing sensitive user data.
- **Error Handling**: The application smoothly manages invalid inputs, upload problems, and internal errors, offering users clear feedback while logging technical details for administrators.

- **Extensibility**: Designed with a modular architecture, the prototype can be easily expanded to include additional features such as phishing source tracing, natural language explanations, or integration with email gateways.

5 Testing and Evaluation

5.1 Testing Scenarios

The phishing detection model was tested on a diverse set of 25 email samples comprising both phishing and non-phishing emails. These samples included:

- Real-world phishing emails with various common phishing tactics (e.g., account alerts, fake invoices, password reset requests)
- Legitimate emails such as event invitations, meeting reminders, and project updates
- Edge cases including ambiguous or borderline emails with mixed or suspicious content

5.2 Performance Metrics

The model's performance was evaluated using the following metrics:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**

5.3 Results

The classification performance of the phishing detection model on the test set of 25 samples is summarized in Table 3.

Table 3: Classification Report on Test Set (25 samples)

Class	**Precision**	**Recall**	**F1-score**	**Support**
Non-phishing	0.67	0.46	0.55	13
Phishing	0.56	0.75	0.64	12
Accuracy		0.60		
Macro avg	0.61	0.61	0.59	25
Weighted avg	0.62	0.60	0.59	25

5.4 Interpretability and Feature Contributions

For each tested email, the model outputs the predicted label along with top contributing features and their impact scores. These feature contributions provide insight into the decision-making process of the classifier, indicating which words or tokens pushed the prediction towards phishing or non-phishing.

Examples include:

- *Lottery scam* emails showed strong positive contributions from words like "000", "claim", and "details," driving the phishing classification.
- Legitimate emails such as *Event invitation* and *Meeting reminder* had dominant negative feature contributions, reflecting trusted terms that reduce phishing likelihood.
- Some false negatives and false positives revealed ambiguous feature weights, highlighting challenges in edge cases (e.g., *Fake invoice attached* predicted as non-phishing).

5.5 Analysis

The evaluation results reveal:

- An overall accuracy of 60% on the small test set.
- Higher recall for the phishing class (0.75) compared to non-phishing (0.46), indicating the model is better at catching phishing emails.
- Precision is higher for non-phishing (0.67) than phishing (0.56), meaning more false positives occur in phishing classification.
- The F1-scores reflect a trade-off between precision and recall, with phishing detection slightly outperforming non-phishing.
- The feature contribution analysis aids interpretability and provides clues for improving model robustness.

5.6 Failure Analysis

Despite demonstrating reasonable performance, the phishing detection system exhibits several failure modes that limit its overall effectiveness:

- **False Positives on Legitimate Emails:** Several non-phishing emails were incorrectly classified as phishing. For example, emails such as *Flight booking confirmation*, *Monthly billing statement*, and *Schedule change notice* were flagged as phishing despite being legitimate. This suggests the model sometimes over-relies on certain negative feature contributions (e.g., words like "claim" and "security") that may appear in legitimate contexts but are associated with phishing in training.

- **False Negatives on Phishing Emails:** Some phishing emails were misclassified as non-phishing, notably *Fake invoice attached* and *Compromised password alert.* This indicates the model can fail to recognize more sophisticated or subtle phishing tactics, especially when key phishing indicators have weaker feature weights or overlap with legitimate language patterns.
- **Ambiguity in Edge Cases:** Emails with mixed content or borderline characteristics (e.g., *Customer feedback request* and *Newsletter subscription*) showed inconsistent classification, reflecting challenges in distinguishing nuanced differences between phishing and legitimate emails.
- **Feature Weight Conflicts:** The top contributing features sometimes include terms with contradictory impacts (both positive and negative scores across samples). This inconsistency suggests that the model's feature weighting may not fully capture contextual usage, leading to erroneous predictions in some cases.

Implications: These failure points highlight the need for:

- Enhanced feature engineering to better capture contextual nuances and reduce false alarms.
- Incorporation of additional data sources or more advanced models (e.g., deep learning or context-aware embeddings) to improve detection of sophisticated phishing.
- Ongoing evaluation with larger, more diverse datasets to identify and mitigate weaknesses.

Resolving these challenges is essential to enhance the model's reliability and make it more effective for real-world phishing detection applications. Overall, the model demonstrates potential in identifying phishing attempts, with a strong emphasis on maximising recall to minimise missed threats. Nonetheless, the moderate precision and recall scores indicate that further improvements are needed, especially to lower false positive rates on legitimate emails.

6 Limitations and Considerations

1. **Feature Limitations**
    - Does not analyse email headers beyond basic sender information.
    - Limited processing of HTML content and complex email structures.
    - No support for image or attachment analysis, limiting detection of embedded malicious content.
    - Only supports plain text input and simple file formats (e.g., .txt), restricting input diversity.
2. **Dataset Biases**
    - Training data may not cover the latest phishing techniques or emerging threats.

- Geographic, cultural, and language biases present in the training dataset may reduce model generalizability.
- Potential imbalance in phishing vs. legitimate emails affecting classifier performance.

3. **Security Considerations**
    - The system is vulnerable to adversarial attacks crafted to evade detection.
    - Regular model retraining and security patching are required to maintain detection efficacy.
    - Input validation and sanitisation are vital, but they may not completely eliminate risks associated with injection or cross-site scripting (XSS).
    - While rate limiting and session management offer fundamental protection, they should be enhanced with more robust mechanisms in production environments.
4. **Performance Trade-offs**
    - Balancing precision and recall is challenging: Optimising for fewer false positives can increase missed phishing attempts.
    - Higher sensitivity may result in more false alarms, potentially overwhelming users.
    - Computational resources and latency considerations may limit real-time analysis capabilities at scale.
5. **Operational and Scalability Considerations**
    - Current implementation uses in-memory rate limiting, which does not scale across multiple server instances.
    - Model and vectoriser loading processes are static and lack versioning or hot-reloading features.
    - Lack of HTTPS enforcement and advanced session security features may expose user data in transit.
    - Limited logging granularity reduces the ability to perform detailed audit trails or incident investigations.

7 References

References

[1] N. Abdullah, “Phishing Email Dataset,” *Kaggle*, 2024. [Online]. Available: `https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset`. [Accessed: May 12, 2025].

[2] PlantUML, “Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams,” *PlantUML.com*, [Online]. Available: `https://plantuml.com/`. [Accessed: May 17, 2025].

[3] Mermaid, “Mermaid — Diagramming and charting tool,” *Mermaid.js.org*, [Online]. Available: `https://mermaid.js.org/`. [Accessed: May 12, 2025].

[4] A. Alhuzali, A. Alloqmani, M. Aljabri, and F. Alharbi, “In-depth analysis of phishing email detection: Evaluating the performance of machine learning and deep learning models across multiple datasets,” *Applied Sciences*, vol. 15, no. 6, p. 3396, Mar. 2025. doi: `https://doi.org/10.3390/app15063396`.

[5] Excalidraw, “Excalidraw,” *Excalidraw*, [Online]. Available: `https://excalidraw.com/`. [Accessed: May 12, 2025].

[6] Scikit-Learn, “sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 documentation,” *Scikit-learn.org*, [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`. [Accessed: May 12, 2025].

[7] Scikit-Learn, “sklearn.linear model.LogisticRegression — scikit-learn 0.21.2 documentation,” *Scikit-learn.org*, 2014. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`. [Accessed: May 12, 2025].

[8] IBM, “What is a Neural Network?,” *IBM*, Oct. 06, 2021. [Online]. Available: `https://www.ibm.com/think/topics/neural-networks`. [Accessed: May 10, 2025].

[9] IBM, “Support Vector Machine,” *IBM*, Dec. 12, 2023. [Online]. Available: `https://www.ibm.com/think/topics/support-vector-machine`. [Accessed: May 10, 2025].

[10] IBM, “What Is a Decision Tree?,” *Ibm.com*, Nov. 2, 2021. [Online]. Available: `https://www.ibm.com/think/topics/decision-trees`

[11] J. Daniel and J. Martin, *Speech and Language Processing*, Dec. 2021. [Online]. Available: `https://web.stanford.edu/~jurafsky/slp3/4.pdf`

[12] C. Staff, “The Difference Between SVM and Decision Trees,” *Coursera*, 2025. [Online]. Available: `https://www.coursera.org/articles/difference-between-svm-and-decision-tree`. [Accessed: April 8, 2025].

[13] “Support Vector Machines vs Neural Networks,” *GeeksforGeeks*, Feb. 13, 2024. [Online]. Available: `https://www.geeksforgeeks.org/support-vector-machines-vs-neural-networks/`.[Accessed: April 8 2025].

[14] Flask, “Welcome to Flask — Flask Documentation (3.0.x),” *Palletsprojects.com*, 2010. [Online]. Available: `https://flask.palletsprojects.com/en/stable/`

[15] Google, “Classification: Accuracy, recall, precision, and related metrics,” *Google for Developers*, 2024. [Online]. Available: `https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall`. [Accessed: May 2025].

[16] Acunetix. *What is Cross-site Scripting and How Can You Fix it?* Acunetix Web Application Security. Available at: `https://www.acunetix.com/websitesecurity/cross-site-scripting/` [Accessed May 17, 2025].