National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Manideep Kokkalakonda
Student ID: x23244488

School of Computing
National College of Ireland

Supervisor: Joel Aleburu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Manideep Kokkalakonda		
Student ID:	X23244488		
Programme:	Msc in Cybersecurity	**Year:**	2024-25
Module:	MSc (Research) Practicum/Internship		
Lecturer:	Joel Aleburu		
Submission Due Date:	15/09/2025		
Project Title:	Phishing Detection and Prevention Using Natural Language Processing (NLP		
Word Count:	1213	**Page Count:**	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Manideep Kokkalakonda

Date: 15/09/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Manideep Kokkalakonda
Student ID: x23244488

1 Introduction

This configuration manual provides step-by-step guidance for installing, configuring, and executing the Hybrid Phishing Email Detection Framework developed for the MSc Research Project.

The system combines traditional machine learning models using TF-IDF features with a BERT + BiLSTM hybrid deep learning model enriched with structural features. It is implemented in Python with support for explainable AI techniques (SHAP and LIME). The manual outlines software and hardware requirements, dataset setup, environment configuration, execution instructions, and troubleshooting tips to ensure reproducibility.

2 System Requirements

2.1 Hardware Requirements

- **Processor:** Minimum Intel Core i5 or equivalent (Quad-core recommended)
- **RAM:** Minimum 8 GB (16 GB recommended for faster deep learning processing)
- **GPU:** NVIDIA GPU with at least 4 GB VRAM (CUDA-enabled) for BERT model acceleration (optional but highly recommended)
- **Storage:** Minimum 10 GB free space for datasets, models, and dependencies

2.2 Software Requirements

- **Operating System:** Windows 10 / Ubuntu 20.04 / macOS 12 or later
- **Python:** Version 3.9+
- **IDE/Notebook:** Jupyter Notebook or Google Colab
- **Required Python Libraries:**
    - pandas
    - numpy
    - scikit-learn
    - torch
    - transformers
    - nltk

- matplotlib
- seaborn
- imbalanced-learn
- shap
- lime
- mlflow
- wordcloud

- **Optional:** CUDA Toolkit 11.x (if using GPU)

3 Environment Setup

This section outlines the steps required to configure the environment for running the phishing email detection system. The framework can be executed either locally or on **Google Colab** (recommended for GPU acceleration).

3.1 Local Environment Setup

Install Python 3.9+: Download and install from the official Python website: https://www.python.org/downloads/

1. Library Imports and Setup

```
# Data manipulation and analysis
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# Natural Language Processing
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer

# Machine Learning
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, classification_report, confusion_matrix,
    roc_curve, precision_recall_curve
)
```

Figure 1 : Library Imports and Setup

4 Data Loading and preprocessing

4.1 Getting the Dataset

1. **Download datasets** from the following public sources (all research-licensed):

- CEAS_08 Phishing Corpus
- Enron Email Dataset
- Ling Spam Dataset
- Nazario Phishing Corpus
- Nigerian Fraud Email Dataset
- SpamAssassin Public Corpus
- Consolidated Phishing_Email Dataset

2. **Save all datasets** into the project's datasets/ directory, ensuring each source retains its original filename.
3. **Verify dataset integrity**:
   - Total combined records: **164,971 emails**
     - **Phishing:** 85,781
     - **Legitimate:** 79,190
   - Final balanced experimental subset: **82,486 emails**
     - **Phishing:** 42,891
     - **Legitimate:** 39,595
4. **Check file encoding** to ensure UTF-8 compatibility before processing.

```
# Dataset files to load
dataset_files = [
    '/content/CEAS_08.csv',
    '/content/Enron.csv',
    '/content/Ling.csv',
    '/content/Nazario.csv',
    '/content/Nigerian_Fraud.csv',
    '/content/SpamAssasin.csv',
    '/content/phishing_email.csv'
]

for file_path in dataset_files:
    try:
        df = pd.read_csv(file_path)
        loaded_datasets[file_path] = df
        print(f"Loaded {file_path.split('/')[-1]}: {df.shape[0]} rows, {df.shape[1]} columns")
        print(f"  Columns: {list(df.columns)}")
    except FileNotFoundError:
        print(f"{file_path.split('/')[-1]} not found")
    except Exception as e:
        print(f"Error loading {file_path.split('/')[-1]}: {str(e)}")

print(f"\nSuccessfully loaded {len(loaded_datasets)} datasets")
```

```
[8]
Loaded CEAS_08.csv: 39154 rows, 7 columns
  Columns: ['sender', 'receiver', 'date', 'subject', 'body', 'label', 'urls']
Loaded Enron.csv: 29767 rows, 3 columns
  Columns: ['subject', 'body', 'label']
Loaded Ling.csv: 2859 rows, 3 columns
  Columns: ['subject', 'body', 'label']
Loaded Nazario.csv: 1565 rows, 7 columns
  Columns: ['sender', 'receiver', 'date', 'subject', 'body', 'urls', 'label']
Loaded Nigerian_Fraud.csv: 3332 rows, 7 columns
  Columns: ['sender', 'receiver', 'date', 'subject', 'body', 'urls', 'label']
Loaded SpamAssasin.csv: 5809 rows, 7 columns
  Columns: ['sender', 'receiver', 'date', 'subject', 'body', 'label', 'urls']
Loaded phishing_email.csv: 82486 rows, 2 columns
  Columns: ['text combined', 'label']
```

Figure 2 : Data Loading

```
[9]
...
================================================
Dataset: /content/CEAS_08.csv
================================================
Shape: (39154, 7)
Columns: ['sender', 'receiver', 'date', 'subject', 'body', 'label', 'urls']

Label distribution:
label
1    21842
0    17312
Name: count, dtype: int64
Label types: [1 0]

Missing values:
receiver    462
subject      28
dtype: int64

First 2 rows:
                              sender                       receiver  \
0  Young Esposito <Young@iworld.de>    user4@gvc.ceas-challenge.cc
1      Mok <ipline's1983@icable.ph>  user2.2@gvc.ceas-challenge.cc

                               date                   subject  \
...
```

Figure 3 : Dataset Exploring

4.2 Preprocessing the Data

Merge Datasets

- Combine all sources into a unified CSV file (phishing_dataset.csv).
- Add a dataset_source column to preserve the origin of each email.
- Concatenate subject and body into a single text_combined field.

```
# Combine all datasets into master dataset
print("Combining all datasets into master dataset...")

all_datasets = []
for filename, df in standardized_datasets.items():
    # Remove rows with empty text
    df_clean = df[df['text_combined'].str.strip() != ''].copy()
    if len(df_clean) > 0:
        all_datasets.append(df_clean)
        print(f"Added {len(df_clean)} rows from {filename}")
]
Combining all datasets into master dataset...
Added 39154 rows from /content/CEAS_08.csv
Added 29767 rows from /content/Enron.csv
Added 2859 rows from /content/Ling.csv
Added 1565 rows from /content/Nazario.csv
Added 3332 rows from /content/Nigerian_Fraud.csv
Added 5809 rows from /content/SpamAssasin.csv
Added 82485 rows from /content/phishing_email.csv
```

Figure 4 : 1. Merge Datasets

```
        master_df = pd.concat(all_datasets, ignore_index=True)
        print(f"\nMaster dataset created:")
        print(f"Total rows: {len(master_df)}")
        print(f"Total features: {len(master_df.columns)}")
        print(f"\nLabel distribution:")
        print(master_df['label'].value_counts())
        print(f"\nDataset source distribution:")
        print(master_df['dataset_source'].value_counts())
    else:
        print("No valid datasets to combine")
        master_df = pd.DataFrame()
```

```
Master dataset created:
Total rows: 164971
Total features: 5

Label distribution:
label
1    85781
0    79190
Name: count, dtype: int64

Dataset source distribution:
```

Figure 5 : Master Data Creating

```
# Plot 1: Label Distribution
plt.figure(figsize=(6, 6))
label_counts = master_df['label'].value_counts()
plt.pie(label_counts.values, labels=['Legitimate (0)', 'Phishing (1)'], autopct='%1.1f%%', startangle=90)
plt.title('Overall Label Distribution')
plt.tight_layout()
plt.show()
```

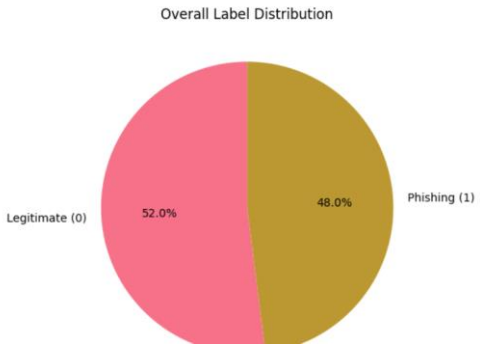


Figure 6 : EDA Overall Label Distribution

```
# Generate enhanced word clouds
if 'label' in main_df.columns:
    fig, axes = plt.subplots(1, 2, figsize=(16, 8))

    # Legitimate emails word cloud
    legitimate_text = ' '.join(main_df[main_df['label'] == 0][text_column].astype(str).values)
    legitimate_wordcloud = WordCloud(
        width=400, height=300,
        background_color='white',
        max_words=100,
        colormap='viridis'
    ).generate(legitimate_text)

    axes[0].imshow(legitimate_wordcloud, interpolation='bilinear')
    axes[0].axis('off')
    axes[0].set_title('Legitimate Emails Word Cloud', fontsize=14, fontweight='bold')

    # Phishing emails word cloud
    phishing_text = ' '.join(main_df[main_df['label'] == 1][text_column].astype(str).values)
    phishing_wordcloud = WordCloud(
        width=400, height=300,
        background_color='white',
        max_words=100,
        colormap='plasma'
    ).generate(phishing_text)
```

Figure 7 : Word Cloud Analysis

```
# Text preprocessing pipeline
print("Setting up text preprocessing...")
```
[24] Python

```
Setting up text preprocessing...
```

```
# Initialize NLTK components
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
```
[25] Python

```
class ImprovedTextPreprocessor:
    """
    Improved Text Preprocessor addressing professor feedback:
    1. Removed stemming (using only lemmatization)
    2. Fixed whitespace loss issues
    3. Better word separation and understanding
    """

    def __init__(self):
        self.lemmatizer = WordNetLemmatizer()
        self.stop_words = set(stopwords.words('english'))

    def remove_html_tags(self, text):
        """Remove HTML tags from text"""
        clean = re.compile('<.*?>')
        return re.sub(clean, ' ', text)  # Replace with space to maintain word separation
```

Figure 8 : Text Preprocessing

```
# Show before/after comparison
print("\n=== BEFORE/AFTER PREPROCESSING COMPARISON ===")
for i in range(3):
    print(f"\nExample {i+1}:")
    print(f"ORIGINAL: {sample_df[text_column].iloc[i][:200]}...")
    print(f"PROCESSED: {sample_df['processed_text'].iloc[i][:200]}...")
```
[29] Python

```
=== BEFORE/AFTER PREPROCESSING COMPARISON ===

Example 1:
ORIGINAL: endangered languages workshop foundation endangered languages pleased announce first workshop entitled steps language rescue take place university york weekend 26 27 july yea
PROCESSED: endangered language workshop foundation endangered language pleased announce first workshop entitled step language rescue take place university york weekend july year prograr

Example 2:
ORIGINAL: claretta claretta_bordersfusemailcom cialis wont break wallet senior fellow faculty member james macgregor burns academy leadership university maryland board directors marylar
PROCESSED: claretta claretta bordersfusemailcom cialis wont break wallet senior fellow faculty member james macgregor burn academy leadership university maryland board director marylan

Example 3:
ORIGINAL: roger upole schkeramsncom kyle rickey wrote im trying get file version executable file network times network quite slow prevent blocking main application decided put code sep
PROCESSED: roger upole schkeramsncom kyle rickey wrote im trying get file version executable file network time network quite slow prevent blocking main application decided put code sep
```

Figure 9 : Before/After Preprocessing Comparison

```
print("=== PROPER DATA SPLITTING ===")
print("Implementing train/validation/test split to avoid data leakage")

# First split: separate test set (20%)
X_temp, X_test, y_temp, y_test = train_test_split(
    sample_df['processed_text'],
    sample_df['label'],
    test_size=0.2,
    random_state=42,
    stratify=sample_df['label']
)

# Second split: separate train and validation from remaining 80%
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp,
    test_size=0.25,  # 0.25 * 0.8 = 0.2 (20% of total for validation)
    random_state=42,
    stratify=y_temp
)

print(f"\nData Split Summary:")
print(f"Training set: {len(X_train)} samples ({len(X_train)/len(sample_df)*100:.1f}%)")
print(f"Validation set: {len(X_val)} samples ({len(X_val)/len(sample_df)*100:.1f}%)")
print(f"Test set: {len(X_test)} samples ({len(X_test)/len(sample_df)*100:.1f}%)")

# Check label distribution in each split
print(f"\nLabel Distribution:")
print(f"Training - Legitimate: {(y_train == 0).sum()}, Phishing: {(y_train == 1).sum()}")
print(f"Validation - Legitimate: {(y_val == 0).sum()}, Phishing: {(y_val == 1).sum()}")
print(f"Test - Legitimate: {(y_test == 0).sum()}, Phishing: {(y_test == 1).sum()}")
```
31] Python

Figure 10 : Data Splitting to Avoid Data Leakage

```python
# TF-IDF Feature Extraction
print("\n=== TF-IDF FEATURE EXTRACTION ===")

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(
    max_features=5000,
    min_df=2,
    max_df=0.95,
    ngram_range=(1, 2),
    use_idf=True,
    stop_words='english'
)
```

[32] Python

```
=== TF-IDF FEATURE EXTRACTION ===
```

```python
# Fit on training data only (CRITICAL: prevents data leakage)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print(f"TF-IDF Features Shape:")
print(f"Training: {X_train_tfidf.shape}")
print(f"Validation: {X_val_tfidf.shape}")
print(f"Test: {X_test_tfidf.shape}")
print(f"Feature names: {len(tfidf_vectorizer.get_feature_names_out())}")
```

[33] Python

```
TF-IDF Features Shape:
Training: (6000, 5000)
Validation: (2000, 5000)
Test: (2000, 5000)
Feature names: 5000
```

Figure 11 : TF-IDF Feature Extraction

```python
print("=== COMPREHENSIVE MODEL TRAINING AND EVALUATION ===")

from sklearn.ensemble import GradientBoostingClassifier

# Define models
models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42)
}
```

Python

```
=== COMPREHENSIVE MODEL TRAINING AND EVALUATION ===
```

```python
# Store results
results = []
trained_models = {}
roc_data = {}
confusion_matrices = {}
```

Python

```python
# Train and evaluate each model
for model_name, model in models.items():
    print(f"\nTraining {model_name}...")

    # Train on training set
    model.fit(X_train_tfidf, y_train)

    # Predictions on validation set
    y_pred = model.predict(X_val_tfidf)
    y_pred_proba = model.predict_proba(X_val_tfidf)[:, 1] if hasattr(model, 'predict_proba') else None
```

Figure 12 : ML Models Training and evaluation

5 Machine Learning Models and Evaluation

```
        'CV_Min': cv_scores.min(),
        'CV_Max': cv_scores.max()
    })

    print(f"  CV F1-Score: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")
    print(f"  Individual scores: {[f'{score:.4f}' for score in cv_scores]}")
```

[42] Python

```
Performing 5-fold cross-validation for Logistic Regression...
  CV F1-Score: 0.9657 (+/- 0.0071)
  Individual scores: ['0.9708', '0.9667', '0.9600', '0.9667', '0.9641']

Performing 5-fold cross-validation for Random Forest...
  CV F1-Score: 0.9577 (+/- 0.0085)
  Individual scores: ['0.9567', '0.9500', '0.9600', '0.9625', '0.9592']

Performing 5-fold cross-validation for Gradient Boosting...
  CV F1-Score: 0.9312 (+/- 0.0164)
  Individual scores: ['0.9316', '0.9156', '0.9382', '0.9332', '0.9374']
```

```
cv_results_df = pd.DataFrame(cv_results)
print("\n=== CROSS-VALIDATION SUMMARY ===")
print(cv_results_df.round(4))
```

[43] Python

```
=== CROSS-VALIDATION SUMMARY ===
                 Model  CV_Mean  CV_Std  CV_Min  CV_Max
0  Logistic Regression   0.9657  0.0036  0.9600  0.9708
1        Random Forest   0.9577  0.0043  0.9500  0.9625
2    Gradient Boosting   0.9312  0.0082  0.9156  0.9382
```

Figure 13 : Five-Fold Cross Validation

```
# Feature importance analysis for applicable models
print("=== FEATURE IMPORTANCE ANALYSIS ===")

# Random Forest feature importance
rf_model = trained_models['Random Forest']
rf_importance = rf_model.feature_importances_
rf_top_idx = np.argsort(rf_importance)[-20:][::-1]

print("\nTop 20 Important Features (Random Forest):")
for i, idx in enumerate(rf_top_idx, 1):
    print(f"{i:2d}. {feature_names[idx]:25} (Importance: {rf_importance[idx]:.4f})")
```

[49]

```
=== FEATURE IMPORTANCE ANALYSIS ===

Top 20 Important Features (Random Forest):
 1. aug                       (Importance: 0.0353)
 2. wrote                     (Importance: 0.0215)
 3. enron                     (Importance: 0.0213)
 4. thanks                    (Importance: 0.0133)
 5. list                      (Importance: 0.0116)
 6. pm                        (Importance: 0.0113)
 7. question                  (Importance: 0.0112)
 8. im                        (Importance: 0.0092)
```

Figure 14 : Feature Importance Analysis

```
# Statistical analysis of text characteristics
print("Statistical Analysis of Text Characteristics")
print("="*50)
```

[47] Python

```
Statistical Analysis of Text Characteristics
==================================================
```

```
# 11. Statistical Analysis and Hypothesis Testing

print("=== STATISTICAL ANALYSIS AND HYPOTHESIS TESTING ===")

# Extract processed text lengths by class
legit_lengths = sample_df[sample_df['label'] == 0]['processed_text'].str.len()
phish_lengths = sample_df[sample_df['label'] == 1]['processed_text'].str.len()

# Summary statistics
print(f"Legitimate emails - Mean: {legit_lengths.mean():.1f}, Median: {legit_lengths.median():.1f}, Std: {legit_lengths.std():.1f}")
print(f"Phishing emails   - Mean: {phish_lengths.mean():.1f}, Median: {phish_lengths.median():.1f}, Std: {phish_lengths.std():.1f}")
```

[48] Python

```
=== STATISTICAL ANALYSIS AND HYPOTHESIS TESTING ===
Legitimate emails - Mean: 1438.1, Median: 711.0, Std: 3402.1
Phishing emails   - Mean: 873.0, Median: 352.0, Std: 3225.7
```

```
# Perform independent t-test
t_stat, p_value = stats.ttest_ind(legit_lengths, phish_lengths, equal_var=False)
print(f"\nT-test for difference in mean processed text length:")
print(f"t-statistic = {t_stat:.4f}, p-value = {p_value:.4e}")

if p_value < 0.05:
    print("Result: Statistically significant difference in processed text length between phishing and legitimate emails (p < 0.05).")
else:
    print("Result: No statistically significant difference in processed text length (p >= 0.05).")
```

[49] Python

```
T-test for difference in mean processed text length:
```

Figure 15 : Statistical Analysis and Hypothesis Testing

6 BERT+BiLSTM Hybrid Architecture Implementation

```python
# Import additional libraries for advanced features
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, TensorDataset
from transformers import AutoTokenizer, AutoModel, AutoConfig
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold
from imblearn.over_sampling import SMOTE
import mlflow
import mlflow.pytorch
import shap
import lime
from lime.lime_text import LimeTextExplainer
import pickle
import os
from datetime import datetime
import json

# Set device for PyTorch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Create directories for organized output
os.makedirs('data/features', exist_ok=True)
os.makedirs('models', exist_ok=True)
os.makedirs('results', exist_ok=True)
os.makedirs('explainability', exist_ok=True)

print("Advanced libraries imported and directories created successfully!")
```

Figure 16 : Import libraries

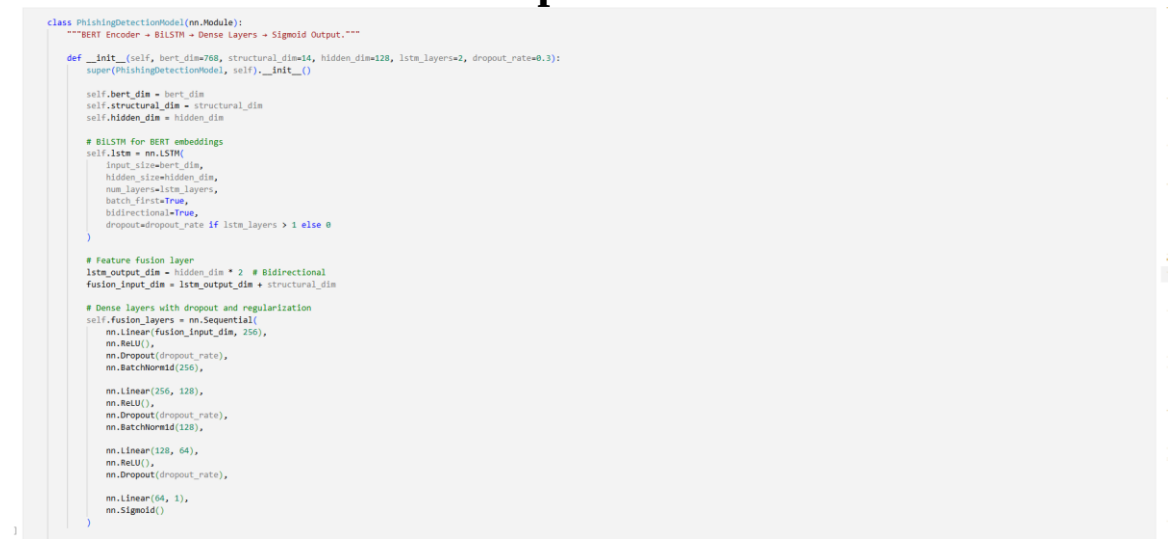
```python
def get_bert_embeddings(texts, batch_size=16, max_length=512):
    """Generate BERT embeddings for a list of texts."""
    embeddings = []

    # Process in batches to manage memory
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]

        # Tokenize batch
        encoded = tokenizer(
            batch_texts,
            padding=True,
            truncation=True,
            max_length=max_length,
            return_tensors='pt'
        )

        # Move to device
        input_ids = encoded['input_ids'].to(device)
        attention_mask = encoded['attention_mask'].to(device)

        # Get embeddings
        with torch.no_grad():
            outputs = bert_model(input_ids=input_ids, attention_mask=attention_mask)
            # Use [CLS] token embedding (first token)
            cls_embeddings = outputs.last_hidden_state[:, 0, :]
            embeddings.extend(cls_embeddings.cpu().numpy())

        # Print progress
        if (i // batch_size + 1) % 10 == 0:
            print(f"Processed {min(i+batch_size, len(texts))}/{len(texts)} texts")

    return np.array(embeddings)

print("BERT embedding function defined!")
```

Figure 17 : Bert Embeddings

```python
def extract_structural_features(texts):
    """Extract structural and stylistic features from text."""
    features = []

    for text in texts:
        # Text length features
        char_count = len(text)
        word_count = len(text.split())
        sentence_count = len(re.findall(r'[.!?]+', text))

        # Stylistic features
        upper_case_count = sum(1 for c in text if c.isupper())
        exclamation_count = text.count('!')
        question_count = text.count('?')
        url_count = len(re.findall(r'http[s]?://(?:[a-zA-Z][0-9][$-_@.&+][!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', text))
        email_count = len(re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text))

        # Special character ratios
        digit_ratio = sum(1 for c in text if c.isdigit()) / max(char_count, 1)
        upper_ratio = upper_case_count / max(char_count, 1)
        special_char_ratio = sum(1 for c in text if not c.isalnum() and not c.isspace()) / max(char_count, 1)

        # Average word length
        avg_word_length = np.mean([len(word) for word in text.split()]) if word_count > 0 else 0

        # Sentiment-related features
        urgent_words = ['urgent', 'immediate', 'asap', 'hurry', 'quick', 'fast', 'now']
        urgent_count = sum(1 for word in urgent_words if word.lower() in text.lower())

        money_words = ['money', 'cash', 'prize', 'win', 'free', 'offer', 'deal', '$', 'dollar']
        money_count = sum(1 for word in money_words if word.lower() in text.lower())

        features.append([
            char_count, word_count, sentence_count, upper_case_count,
            exclamation_count, question_count, url_count, email_count,
            digit_ratio, upper_ratio, special_char_ratio, avg_word_length,
            urgent_count, money_count
```

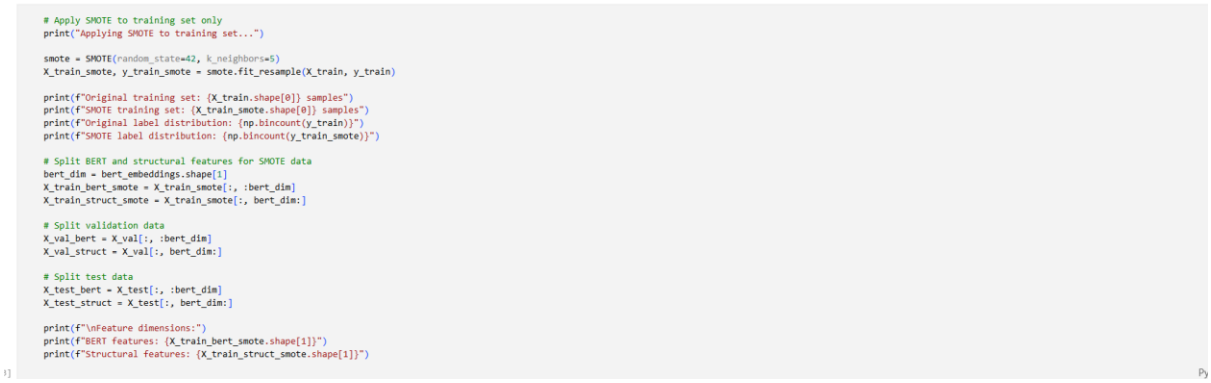
Figure 18 : Extract Structural Features

7 Advanced Model Development with BERT + BiLSTM



```python
class PhishingDetectionModel(nn.Module):
    """BERT Encoder → BiLSTM → Dense Layers → Sigmoid Output."""

    def __init__(self, bert_dim=768, structural_dim=14, hidden_dim=128, lstm_layers=2, dropout_rate=0.3):
        super(PhishingDetectionModel, self).__init__()

        self.bert_dim = bert_dim
        self.structural_dim = structural_dim
        self.hidden_dim = hidden_dim

        # BiLSTM for BERT embeddings
        self.lstm = nn.LSTM(
            input_size=bert_dim,
            hidden_size=hidden_dim,
            num_layers=lstm_layers,
            batch_first=True,
            bidirectional=True,
            dropout=dropout_rate if lstm_layers > 1 else 0
        )

        # Feature fusion layer
        lstm_output_dim = hidden_dim * 2  # Bidirectional
        fusion_input_dim = lstm_output_dim + structural_dim

        # Dense layers with dropout and regularization
        self.fusion_layers = nn.Sequential(
            nn.Linear(fusion_input_dim, 256),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.BatchNorm1d(256),

            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.BatchNorm1d(128),

            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(dropout_rate),

            nn.Linear(64, 1),
            nn.Sigmoid()
        )
```

Figure 19 : Define Model Architecture



```python
# Apply SMOTE to training set only
print("Applying SMOTE to training set...")

smote = SMOTE(random_state=42, k_neighbors=5)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print(f"Original training set: {X_train.shape[0]} samples")
print(f"SMOTE training set: {X_train_smote.shape[0]} samples")
print(f"Original label distribution: {np.bincount(y_train)}")
print(f"SMOTE label distribution: {np.bincount(y_train_smote)}")

# Split BERT and structural features for SMOTE data
bert_dim = bert_embeddings.shape[1]
X_train_bert_smote = X_train_smote[:, :bert_dim]
X_train_struct_smote = X_train_smote[:, bert_dim:]

# Split validation data
X_val_bert = X_val[:, :bert_dim]
X_val_struct = X_val[:, bert_dim:]

# Split test data
X_test_bert = X_test[:, :bert_dim]
X_test_struct = X_test[:, bert_dim:]

print(f"\nFeature dimensions:")
print(f"BERT features: {X_train_bert_smote.shape[1]}")
print(f"Structural features: {X_train_struct_smote.shape[1]}")
```

Figure 20 : Applying Smote

```python
# Initialize MLflow
mlflow.set_experiment("Phishing_Detection_BERT_BiLSTM")

def train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, config):
    """Train the model with early stopping and logging."""

    with mlflow.start_run():
        # Log hyperparameters
        mlflow.log_params(config)
        mlflow.log_param("model_type", "BERT_BiLSTM")
        mlflow.log_param("device", str(device))

        best_val_loss = float('inf')
        patience_counter = 0
        best_model_state = None

        train_losses = []
        val_losses = []
        val_accuracies = []

        for epoch in range(config['epochs']):
            # Training phase
            model.train()
            train_loss = 0.0
            train_correct = 0
            train_total = 0

            for batch_bert, batch_struct, batch_labels in train_loader:
                optimizer.zero_grad()

                # Forward pass
                outputs = model(batch_bert, batch_struct)
                loss = criterion(outputs, batch_labels)

                # Backward pass
                loss.backward()

                # Gradient clipping
                torch.nn.utils.clip_grad_norm_(model.parameters(), config['grad_clip'])

                optimizer.step()

                # Statistics
```

Figure 21 : Training Model with early stop

```python
# Evaluate on test set
def evaluate_model(model, X_bert, X_struct, y_true, batch_size=32):
    """Evaluate model and return predictions and probabilities."""
    model.eval()

    # Convert to tensors
    X_bert_tensor = torch.FloatTensor(X_bert).to(device)
    X_struct_tensor = torch.FloatTensor(X_struct).to(device)

    all_predictions = []
    all_probabilities = []

    # Process in batches
    with torch.no_grad():
        for i in range(0, len(X_bert), batch_size):
            end_idx = min(i + batch_size, len(X_bert))
            batch_bert = X_bert_tensor[i:end_idx]
            batch_struct = X_struct_tensor[i:end_idx]

            # Get predictions
            outputs = model(batch_bert, batch_struct)
            probabilities = outputs.cpu().numpy()
            predictions = (probabilities > 0.5).astype(int)

            all_predictions.extend(predictions)
            all_probabilities.extend(probabilities)

    return np.array(all_predictions), np.array(all_probabilities)

# Evaluate on test set
print("Evaluating on test set...")
test_predictions, test_probabilities = evaluate_model(
    trained_model, X_test_bert, X_test_struct, y_test
)

print(f"Test set evaluation completed!")
print(f"Predictions shape: {test_predictions.shape}")
print(f"Probabilities shape: {test_probabilities.shape}")
```

[77]

Figure 22 : Advnce evaluation

```python
# Perform 5-fold cross-validation on training data
from sklearn.model_selection import StratifiedKFold

def cross_validate_model(X_bert, X_struct, y, n_folds=5):
    """Perform stratified k-fold cross-validation."""

    skf = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)
    cv_results = {
        'accuracy': [],
        'precision': [],
        'recall': [],
        'f1_score': [],
        'roc_auc': []
    }

    print(f"Performing {n_folds}-fold cross-validation...")

    for fold, (train_idx, val_idx) in enumerate(skf.split(X_bert, y)):
        print(f"\nFold {fold + 1}/{n_folds}")

        # Split data
        X_bert_train, X_bert_val = X_bert[train_idx], X_bert[val_idx]
        X_struct_train, X_struct_val = X_struct[train_idx], X_struct[val_idx]
        y_train_fold, y_val_fold = y[train_idx], y[val_idx]

        # Apply SMOTE to training fold
        X_combined_train = np.concatenate([X_bert_train, X_struct_train], axis=1)
        X_combined_smote, y_train_smote = smote.fit_resample(X_combined_train, y_train_fold)

        # Split back
        bert_dim = X_bert_train.shape[1]
        X_bert_train_smote = X_combined_smote[:, :bert_dim]
        X_struct_train_smote = X_combined_smote[:, bert_dim:]

        # Create new model for this fold
        fold_model = PhishingDetectionModel(
            bert_dim=X_bert_train.shape[1],
            structural_dim=X_struct_train.shape[1],
            hidden_dim=128,
            lstm_layers=2,
            dropout_rate=0.3
        ).to(device)

        # Create data loaders
```

Figure 23 : Five-Fold Cross Validation on training data

```python
    plt.tight_layout()
    plt.savefig('results/evaluation_results.png', dpi=300, bbox_inches='tight')
    plt.show()

    print("Comprehensive evaluation plots saved to: results/evaluation_results.png")
```

Python

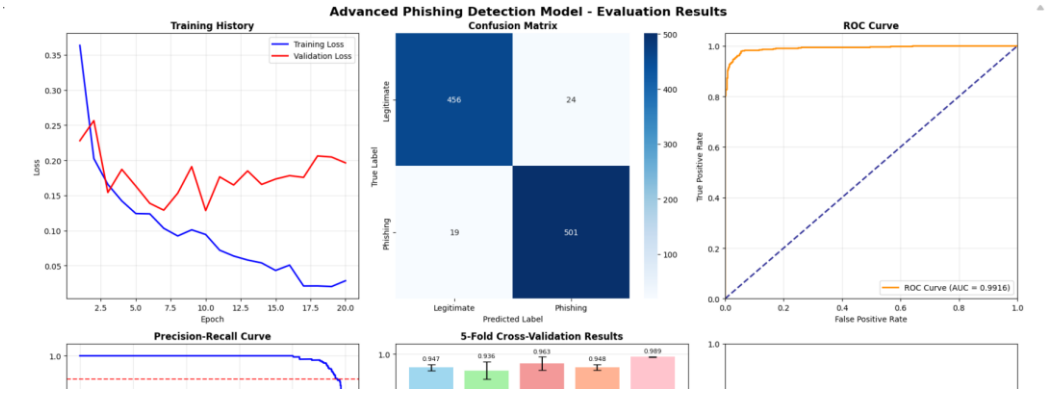


Figure 24 : Advanced Phishing Detection Model - Evaluation Results

8 Explainability & Interpretability with SHAP and LIME

```
# Prepare data for SHAP analysis
# Use a subset for SHAP due to computational constraints
shap_sample_size = min(100, len(X_test))
shap_indices = np.random.choice(len(X_test), shap_sample_size, replace=False)

X_test_shap_bert = X_test_bert[shap_indices]
X_test_shap_struct = X_test_struct[shap_indices]
y_test_shap = y_test[shap_indices]

print(f"Selected {shap_sample_size} samples for SHAP analysis")

# Create a wrapper function for SHAP
def model_predict_for_shap(combined_features):
    """Wrapper function for SHAP that takes combined features."""
    bert_dim = bert_embeddings.shape[1]
    bert_features = combined_features[:, :bert_dim]
    struct_features = combined_features[:, bert_dim:]

    # Convert to tensors
    bert_tensor = torch.FloatTensor(bert_features).to(device)
    struct_tensor = torch.FloatTensor(struct_features).to(device)

    trained_model.eval()
    with torch.no_grad():
        outputs = trained_model(bert_tensor, struct_tensor)
        probabilities = outputs.cpu().numpy()

    # Return both class probabilities for SHAP
    return np.column_stack([1 - probabilities, probabilities])

# Combine test features for SHAP
X_test_combined_shap = np.concatenate([X_test_shap_bert, X_test_shap_struct], axis=1)

# Create SHAP explainer
print("Creating SHAP explainer...")
background_data = X_test_combined_shap[:20]  # Use subset as background
explainer = shap.KernelExplainer(model_predict_for_shap, background_data)

# Calculate SHAP values
print("Calculating SHAP values...")
shap_values = explainer.shap_values(X_test_combined_shap[:20])  # Analyze subset

print(f"SHAP values calculated!")
print(f"SHAP values shape: {np.array(shap_values).shape}")
```

Figure 25 : SHAP Analysis for Global Feature Importance

```
# === LIME WIRING FOR BERT+BiLSTM+STRUCTURAL MODEL ===
from lime.lime_text import LimeTextExplainer
from functools import lru_cache

# Safety checks for objects created earlier
assert 'trained_model' in globals(), "trained_model not found. Train or load your model first."
assert 'tokenizer' in globals(), "tokenizer (HuggingFace) not found."
assert 'bert_model' in globals(), "bert_model (HuggingFace) not found."
assert 'scaler' in globals(), "structural features scaler not found."
assert 'extract_structural_features' in globals(), "extract_structural_features() not found."

trained_model.eval()

# A lighter max_length speeds up LIME a lot (fewer tokens per perturbed sample)
_LIME_MAX_LEN = 256

@torch.no_grad()
def _embed_texts_with_bert(texts, max_length=_LIME_MAX_LEN, batch_size=16):
    """Compute CLS embeddings for a list of texts (numpy array, shape [n, 768])."""
    out = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]
        enc = tokenizer(
            batch, padding=True, truncation=True, max_length=max_length,
            return_tensors='pt'
        )
        input_ids = enc['input_ids'].to(device)
        attention_mask = enc['attention_mask'].to(device)
        outputs = bert_model(input_ids=input_ids, attention_mask=attention_mask)
        cls = outputs.last_hidden_state[:, 0, :].detach().cpu().numpy()
        out.append(cls)
    return np.vstack(out) if out else np.zeros((0, bert_embeddings.shape[1]), dtype=np.float32)

def _struct_features_scaled(texts):
    """Compute structural features and apply the same StandardScaler used in training."""
    feats = extract_structural_features(texts)
    return scaler.transform(feats)

def model_predict_proba_for_lime(texts):
    """
    LIME expects proba for both classes.
```

Figure 26 : LIME Analysis for Local Explanations

```
sample_texts = sample_df['text_combined'].iloc[shap_indices[:5]].tolist()
sample_labels = y_test_shap[:5]
sample_predictions = test_predictions[shap_indices[:5]]
sample_probabilities = test_probabilities[shap_indices[:5]]

lime_explanations = []

print("Generating LIME explanations for sample emails...")

for i, (text, true_label, pred_label, prob) in enumerate(zip(sample_texts, sample_labels, sample_predictions, sample_probabilities)):
    print(f"\nAnalyzing email {i+1}/5...")
    print(f"True label: {true_label}, Predicted: {pred_label}, Probability: {prob:.4f}")

    # Generate LIME explanation
    explanation = lime_explainer.explain_instance(
        text, model_predict_proba_for_lime, num_features=10, num_samples=100
    )

    # Save explanation as HTML
    explanation.save_to_file(f'explainability/lime_explanation_email_{i+1}.html')

    # Extract explanation data with proper type conversion for JSON serialization
    explanation_data = {
        'email_id': int(i + 1),
        'true_label': int(true_label),  # Convert numpy int to Python int
        'predicted_label': int(pred_label),  # Convert numpy int to Python int
        'prediction_probability': float(prob),  # Convert numpy float to Python float
        'text_preview': str(text[:200] + '...' if len(text) > 200 else text),
        'explanation_features': [(str(feature), float(weight)) for feature, weight in explanation.as_list()],  # Convert to JSON-serializable format
        'classification_result': 'Correct' if int(true_label) == int(pred_label) else 'Misclassified',
        'confidence_level': 'High' if abs(float(prob) - 0.5) > 0.4 else 'Medium' if abs(float(prob) - 0.5) > 0.2 else 'Low'
    }

    lime_explanations.append(explanation_data)

    print(f"Top contributing features:")
    for feature, weight in explanation.as_list()[:5]:
        direction = "→ Phishing" if weight > 0 else "→ Legitimate"
        print(f"  '{feature}': {weight:.4f} {direction}")

# Save LIME explanations with proper JSON serialization
try:
    with open('explainability/lime_explanations.json', 'w') as f:
        json.dump(lime_explanations, f, indent=2, ensure_ascii=False)
    print(f"\n✓ LIME explanations saved successfully!")
```

Figure 27 : Analyze specific examples with LIME

9 Ml Model Comparison

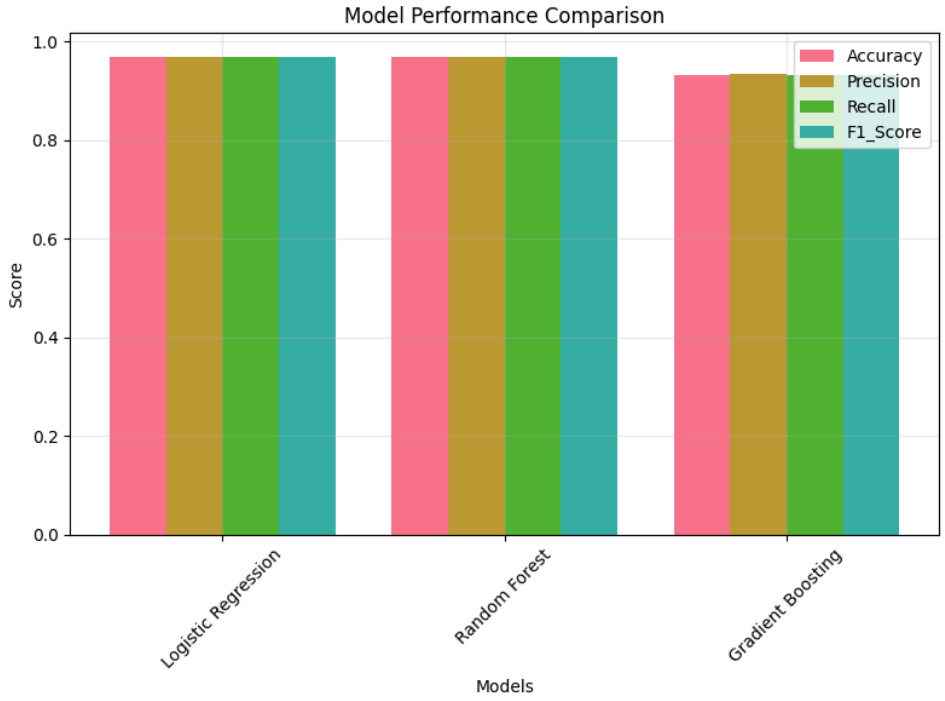


Figure 28 : Model Performance Comprison

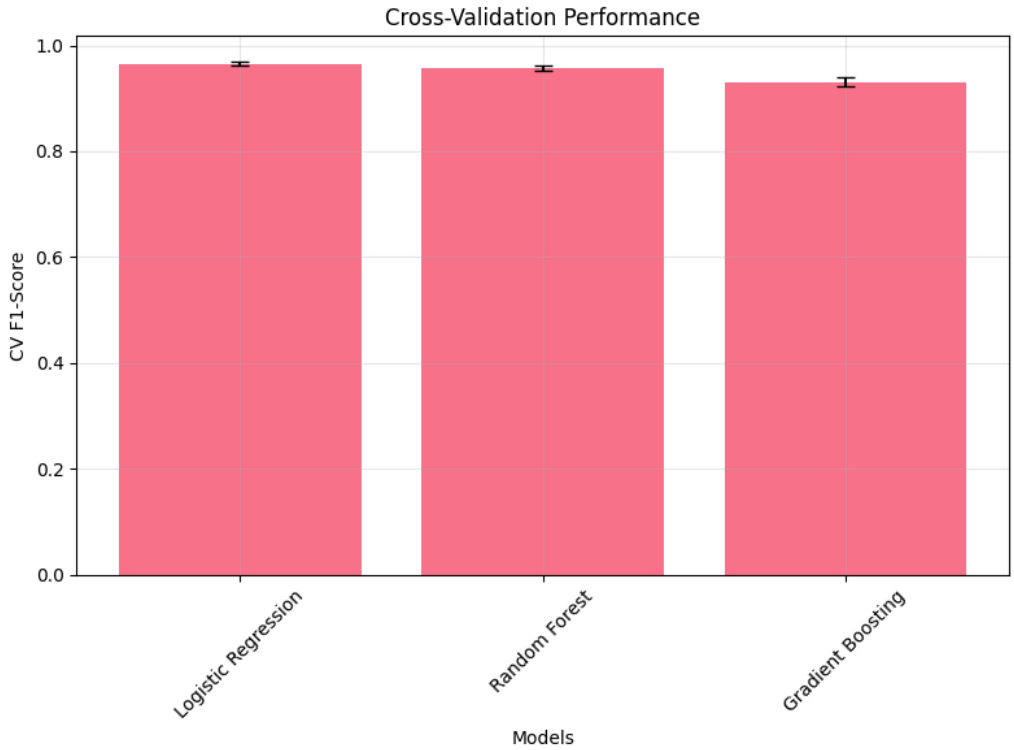


Figure 29 : Cross Validation Performance

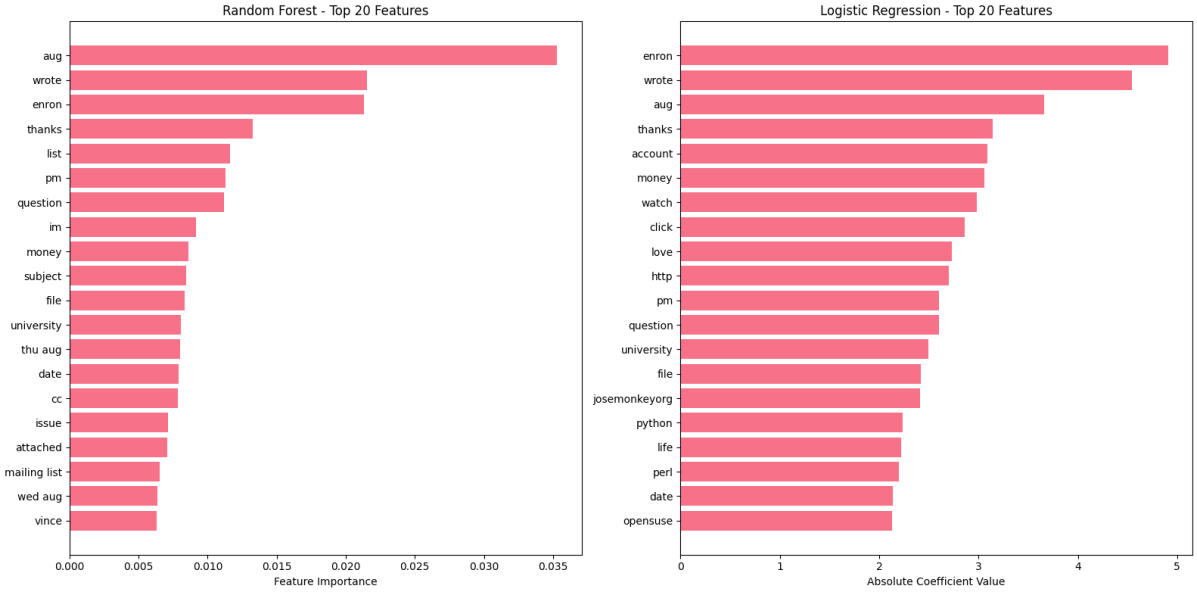


Figure 30 : Top 20 Features of Random Forest and Logic Regression

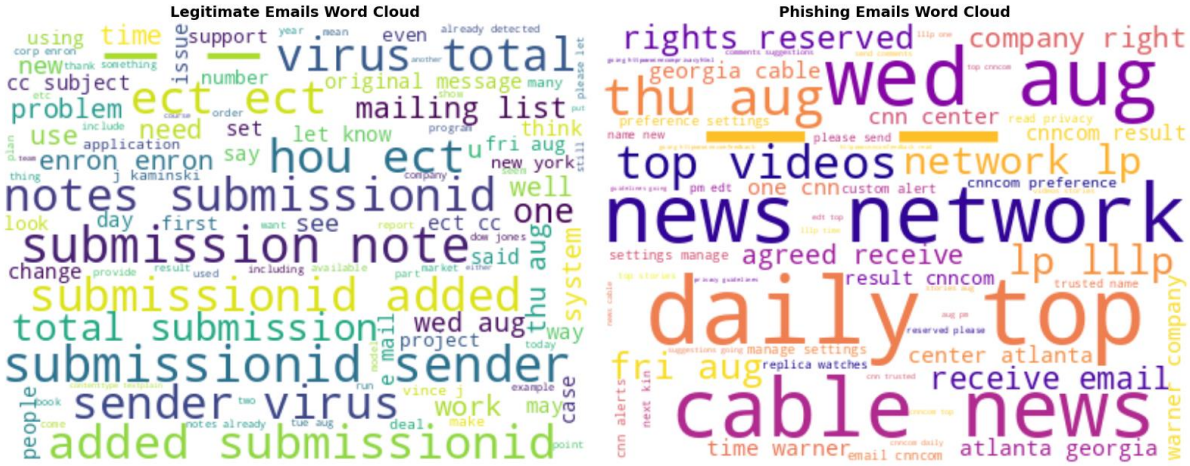


Figure 31 : Legitimate and Phishing Emails word Cloud's

10 Troubleshooting

- **Missing Dependencies:** Ensure all Python libraries from the requirements list are installed with the correct versions (pip install -r requirements.txt).
- **Dataset Issues:** Verify the phishing email dataset files are in the correct directory and match the expected CSV structure.
- **Encoding Errors:** Use encoding='latin-1' when loading CSV files if UnicodeDecodeError occurs.
- **GPU Not Detected:** Ensure NVIDIA drivers and CUDA toolkit are installed and compatible with your PyTorch version.

- **Memory Errors:** Reduce BERT batch size or sequence length to fit available RAM/VRAM.
- **SHAP/LIME Output Errors:** Update the shap and lime packages, and restart the kernel or runtime before rerunning.

References

Abdullah Alam, N. (2024). *Phishing Email Dataset*. [online] www.kaggle.com. Available at: https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. [online] ArXiv. Available at: https://arxiv.org/abs/1810.04805.

Fernández Rodríguez, J., Papale, M., Carminati, M. and Zanero, S. (2022). *A Natural Language Processing Approach for Financial Fraud Detection*. [online] Available at: https://re.public.polimi.it/bitstream/11311/1224432/1/paper10.pdf.

Mckinney, W. (2011). *pandas: a Foundational Python Library for Data Analysis and Statistics*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/265194455_pandas_a_Foundational_Python_Library_for_Data_Analysis_and_Statistics.

Mohammad Amaz Uddin and Sarker, I.H. (2024). An Explainable Transformer-Based Model for Phishing Email Detection: A Large Language Model Approach. [online] doi:https://doi.org/10.2139/ssrn.4785953.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L. and Bai, J. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. [online] arXiv.org. Available at: https://arxiv.org/abs/1912.01703.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É. (2018). Scikit-learn: Machine Learning in Python. *arXiv:1201.0490 [cs]*. [online] Available at: https://arxiv.org/abs/1201.0490.