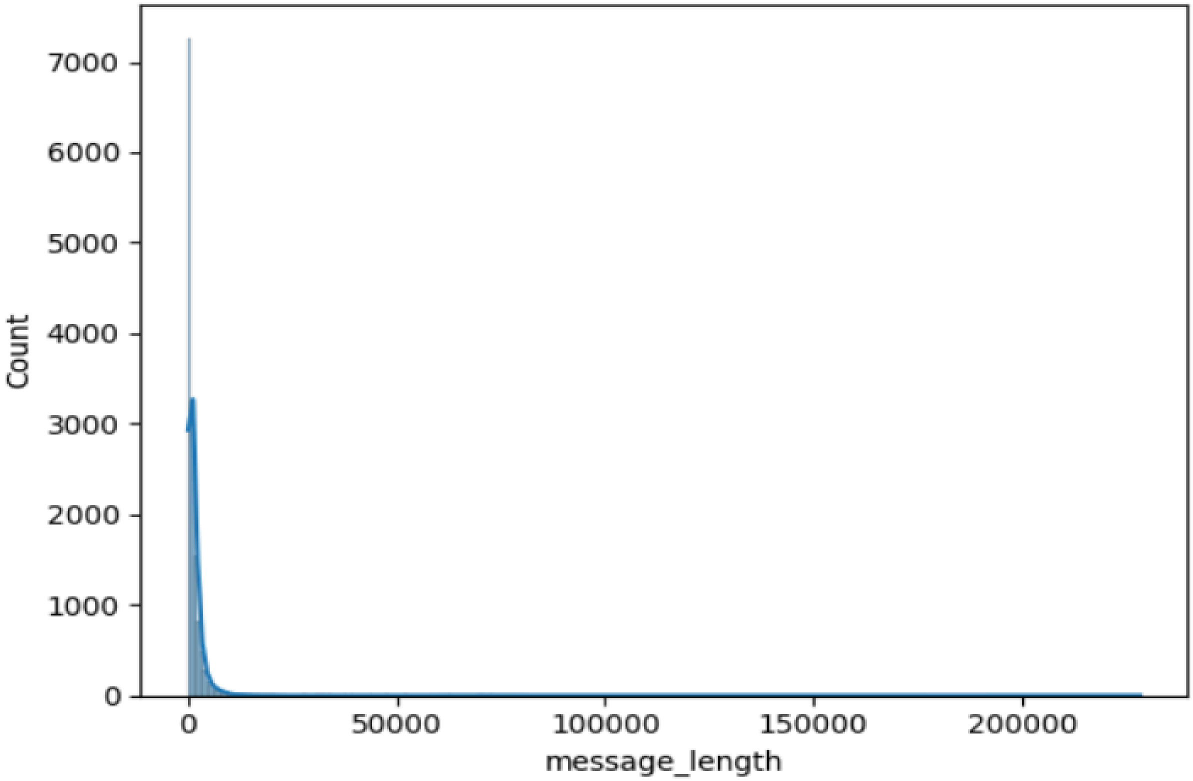
AEGIS SECURE — Model Architecture Explanation

The architecture follows a modular approach, where input data—consisting of message content, metadata, and hyperlinks—is processed through specialized models. These models generate individual outputs that are aggregated in a combination layer to produce a final scam detection score. Additional components, such as an explanation layer and analytics dashboard, enhance interpretability, usability, and monitoring.

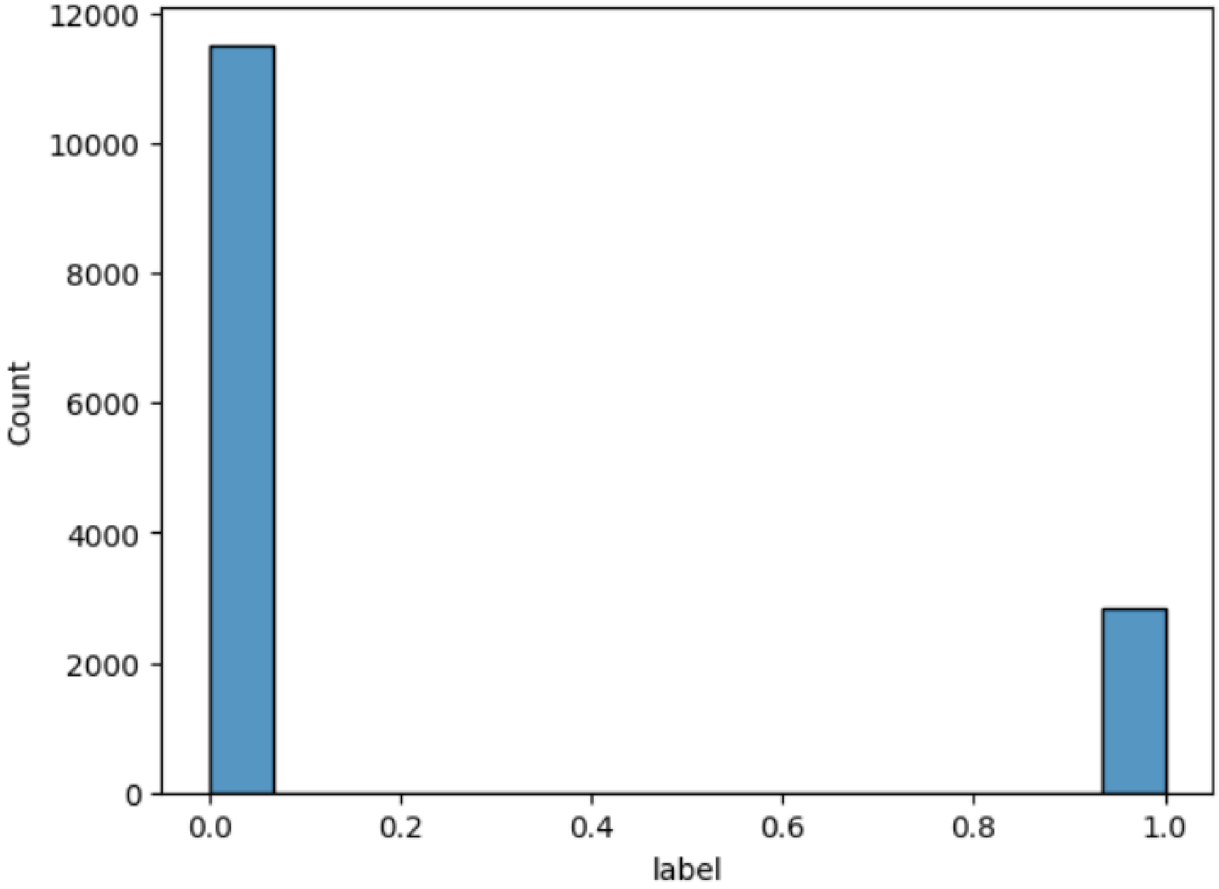
Currently, the team has implemented the **message detection model**, which classifies text as scam or non-scam using a neural network. In the future, the system will be extended to include metadata and hyperlink analysis, explanation layers, and cloud deployment via FastAPI. Initially, the team chose **PyTorch** for its ease of use and rapid prototyping capabilities due to uncertainty about the optimal framework for the project. However, to better support future mobile deployments, the project may use **TensorFlow**, which offers superior optimization for mobile environments through TensorFlow Lite, ensuring efficient performance on resource-constrained devices.

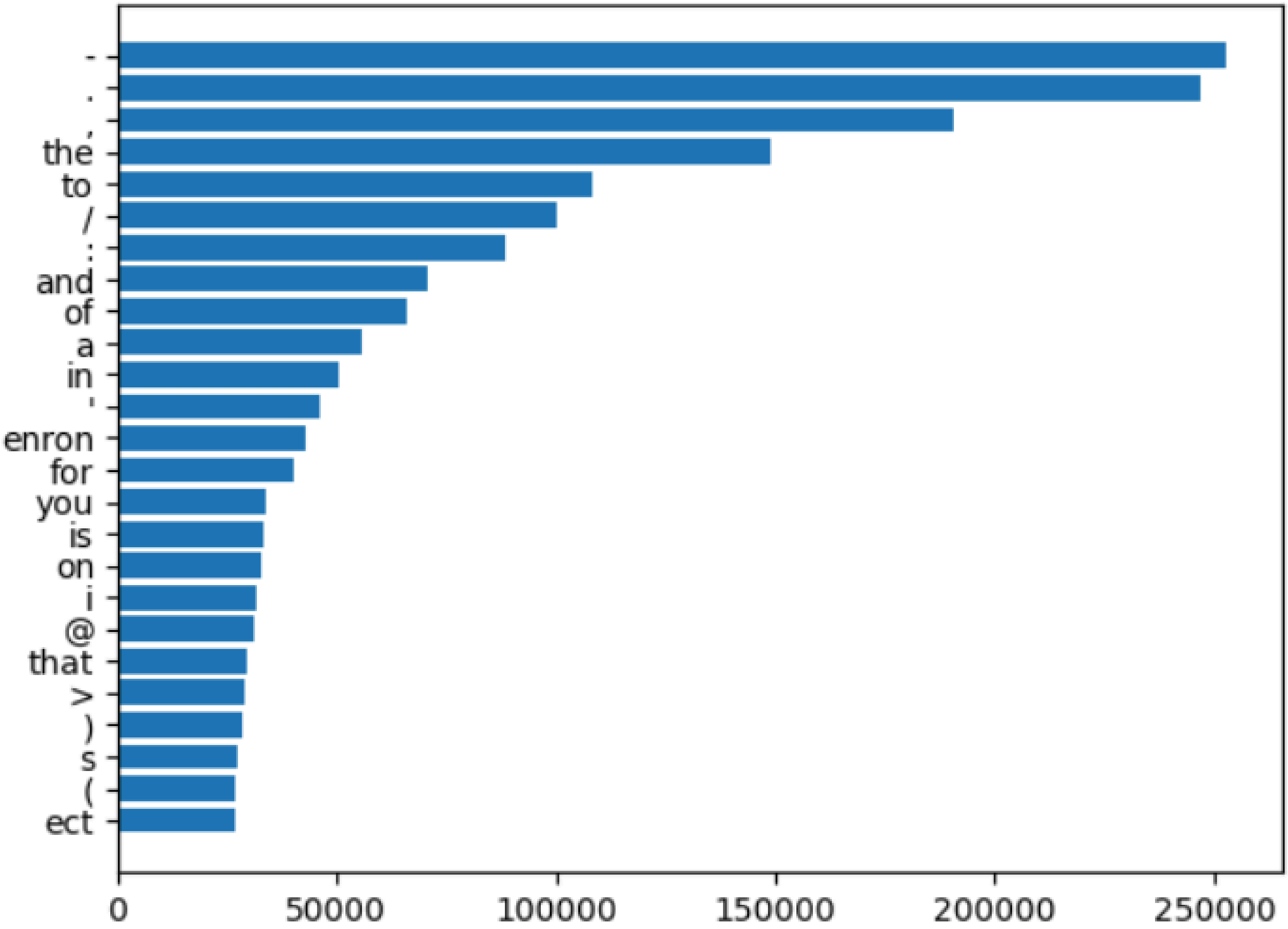
Below is a detailed explanation of the current message detection model and its integration into the broader AEGIS SECURE architecture, with a focus on the role and justification of each library used in the provided code.

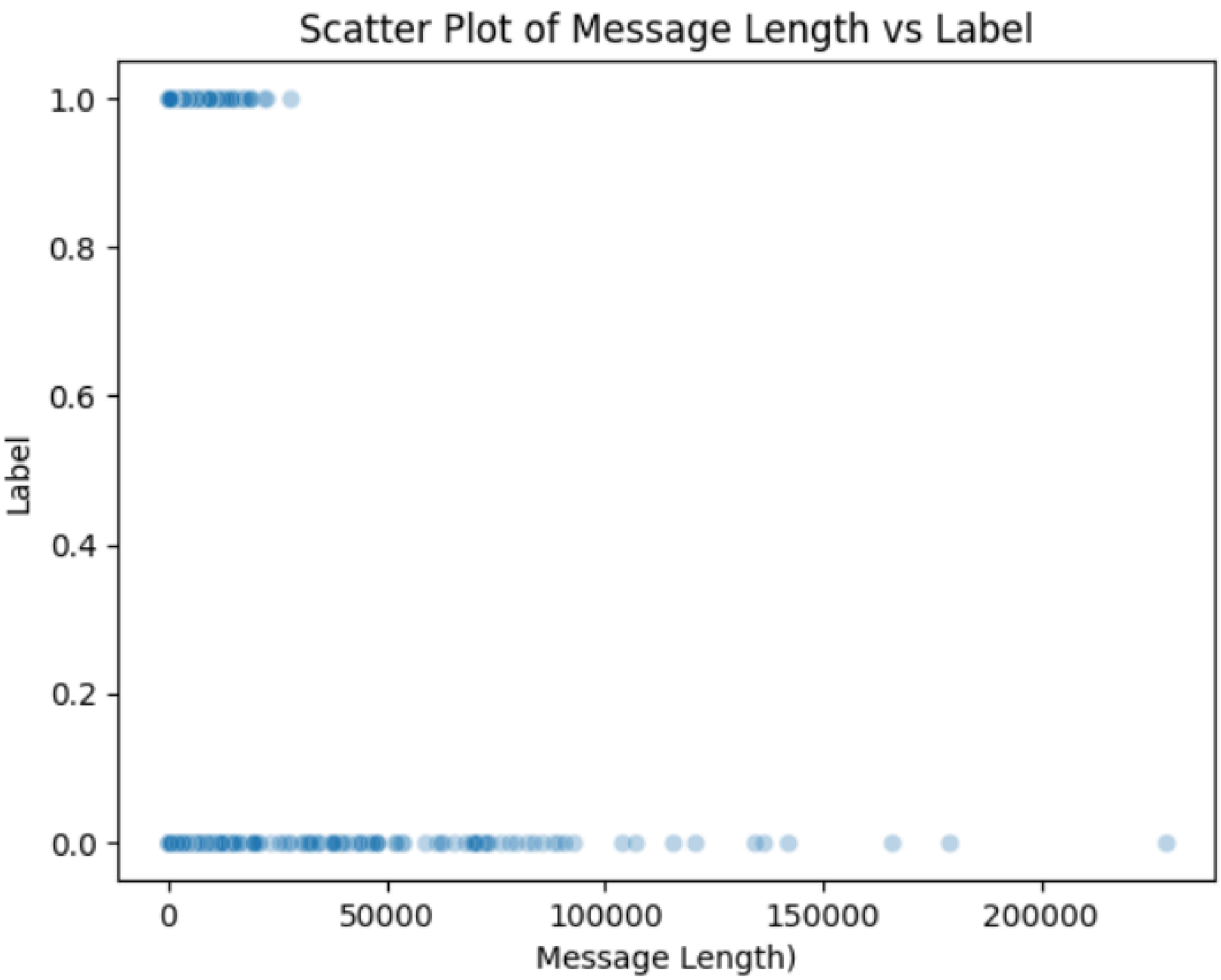
Input Data Statistics:



7000
6000
5000
4000
3000
2000
1000
0
Count
0
50000
100000
150000
200000
message_length


12000
10000
8000
6000
4000
2000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
label


-
.
,
the
to
/
:
and
of
a
in
'
enron
for
you
is
on
i
@
that
>
)
s
(
ect
0
50000
100000
150000
200000
250000



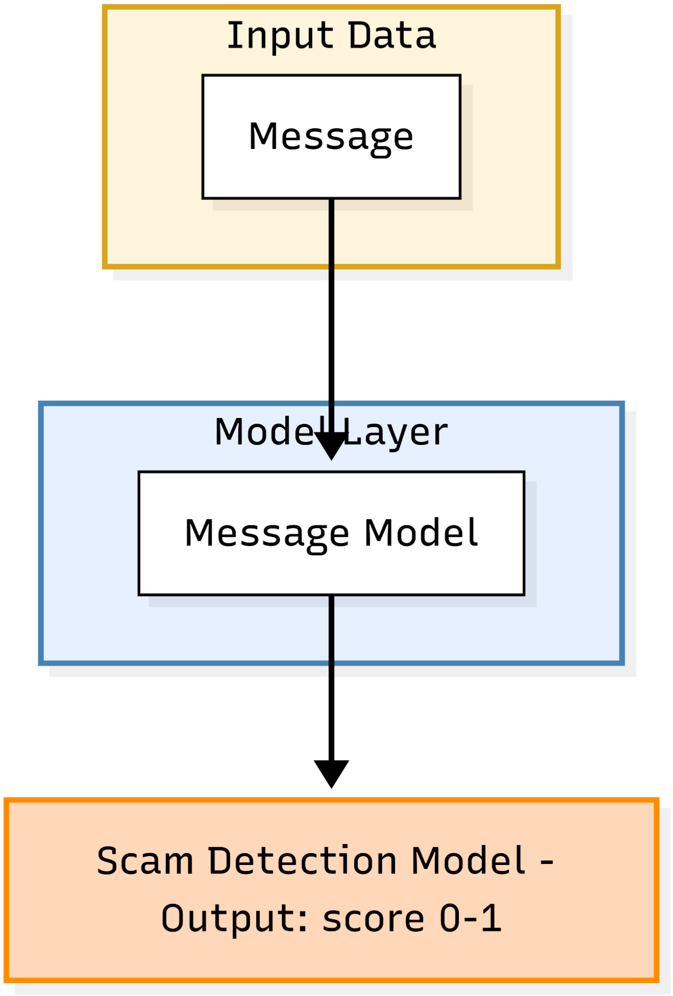
Input Data

1. **Message Content**
   - Description: Raw text from SMS or email, provided via manual input or automated integration.
   - Example: "Your account is compromised, click here to reset your password."
   - Role in Current Implementation: The message detection model processes this text to classify it as scam (1) or safe (0).
   - Future Role: Will be combined with metadata and hyperlink inputs for comprehensive analysis.
2. **Metadata**
   - Description: Attributes such as sender email address, domain, timestamp, or device-specific indicators.
   - Example: Suspicious domain endings (e.g., .xyz) or unusual send times.
   - Future Role: A dedicated metadata model will analyze these attributes to enhance detection accuracy.
3. **Hyperlinks**

- Description: URLs or clickable links embedded in messages.
- Example: Shortened links (e.g., bit.ly) or deceptive domains (e.g., paypal-secure-login.com).
- Future Role: A link model will classify URLs as safe or unsafe to identify phishing attempts.

Rationale: Combining multiple input types ensures a holistic approach to scam detection, capturing textual, contextual, and structural indicators of fraud. The current focus on message content establishes a foundational model, with plans to integrate additional inputs in the future.

Current Model Architecture



Description: Diagram illustrating the current message detection model, showing the flow from raw text input to TF-IDF vectorization, neural network classification, and output scam probability.

The current model focuses on text-based scam detection, using a pipeline that transforms raw message content into numerical features and processes them through a neural network.

Model Layer

The model layer currently consists of the **Data Model** for text analysis, with plans to add **Metadata Model** and **Link Model** in the future.

1. Data Model (Message Text Analysis)

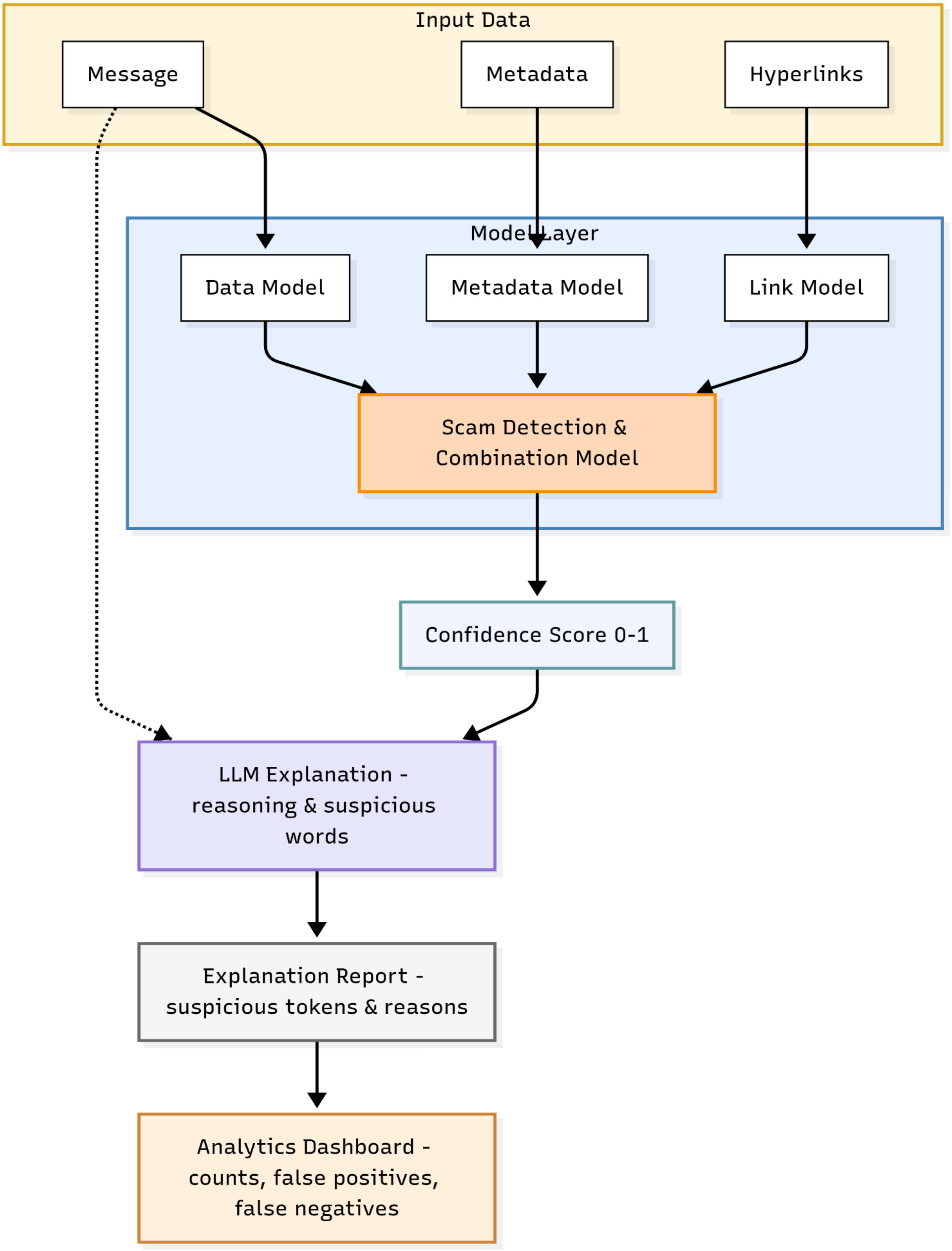
- **Role**: Processes raw text using TF-IDF (Term Frequency–Inverse Document Frequency) vectorization, followed by a deep learning classifier to predict scam likelihood.
- **Implementation (Current Code)**:
    - **Dataset Class (**ScamDataset**)**: Converts TF-IDF sparse matrices and labels into PyTorch-compatible tensors for training. The __getitem__ method transforms sparse rows to dense arrays using .toarray().squeeze().astype('float32') and returns them as torch.tensor objects.
    - **Model Class (**ScamClassifier**)**: A feed-forward neural network (MLP) with seven layers: an input layer (dimension equal to TF-IDF features), five hidden layers (128→256→256→128→64→32 units), and an output layer with a sigmoid activation for binary classification (scam: 1, safe: 0).
    - **Training Loop**: Uses BCELoss (Binary Cross Entropy Loss) and the Adam optimizer to train the model over multiple epochs, with loss printed per epoch.
    - **Evaluation**: Generates predictions on a combined dataset (train+test) and saves true/predicted labels to true.csv and pred.csv for analysis.
    - **Model Saving**: Stores the model's state dictionary and TF-IDF vectorizer in a .pth file for inference.

- **Libraries**:
    - **scikit-learn (**TfidfVectorizer**,** train_test_split**)**:
        - **Purpose**: TfidfVectorizer converts text into numerical features by assigning weights to words based on their frequency and rarity across the dataset. train_test_split splits data into training (60%) and testing (40%) sets for model evaluation.
        - **Why**: TF-IDF is computationally efficient and effective for capturing key terms (e.g., "urgent", "free") indicative of scams. train_test_split ensures

robust evaluation by separating training and testing data, preventing overfitting.

  - **torch and torch.nn**:
    - **Purpose**: Defines the neural network architecture (ScamClassifier) with linear layers, ReLU activations, and dropout (0.4, 0.4, 0.2, 0.2, 0.1) for regularization to prevent overfitting.
    - **Why**: PyTorch's dynamic computation graph and intuitive API simplify model prototyping and experimentation, making it ideal for the current implementation. Its flexibility supports custom layers and training loops, allowing rapid iteration during early development.
  - **torch.optim**:
    - **Purpose**: Implements the Adam optimizer to update model weights based on gradients computed from BCELoss.
    - **Why**: Adam is robust and adaptive, converging faster than traditional gradient descent for deep learning tasks, which is critical for efficient training.
  - **pandas**:
    - **Purpose**: Loads and preprocesses the CSV dataset (with text and label columns), maps labels to binary values (spam/scam → 1, ham → 0), and removes missing data with dropna(). Also saves evaluation results to CSV files.
    - **Why**: Pandas provides efficient data manipulation and cleaning, essential for preprocessing raw text and labels before feeding them into the model.
  - **os and dotenv**:
    - **Purpose**: Manages environment variables (e.g., DATA_PATH, BATCH_SIZE, EPOCHS, LEARNING_RATE, MODEL_DIR, MAX_FEATURES) to configure the training pipeline.
    - **Why**: Using dotenv ensures portability across development and deployment environments, keeping sensitive paths and hyperparameters out of the codebase for security and flexibility.
  - **numpy**:
    - **Purpose**: Handles numerical operations, such as converting sparse TF-IDF matrices to dense arrays and processing evaluation outputs.
    - **Why**: Numpy's efficient array operations are critical for preprocessing and evaluation tasks, especially when interfacing between scikit-learn and PyTorch.
- **Why**: Text is the primary indicator of scams, and TF-IDF with a neural network balances simplicity and expressiveness. PyTorch was chosen for its ease of use during prototyping, as the team was initially uncertain about the optimal framework. However, for future mobile deployments, we plan to transition to **TensorFlow**, which offers better optimization for mobile devices through TensorFlow Lite, enabling lightweight and efficient inference on resource-constrained platforms.

Planned Architecture for future


Input Data
Message
Metadata
Hyperlinks
Model Layer
Data Model
Metadata Model
Link Model
Scam Detection & Combination Model
Confidence Score 0-1
LLM Explanation - reasoning & suspicious words
Explanation Report - suspicious tokens & reasons
Analytics Dashboard - counts, false positives, false negatives

Description: Diagram illustrating the expanded architecture, including the Data Model, Metadata Model, Link Model, Combination Model, LLM Explanation Layer, and Analytics Dashboard, integrated with a FastAPI-based API layer for cloud deployment.

The future implementation will extend the architecture to incorporate additional models and features:

2. Metadata Model

- **Role**: Analyzes contextual attributes (e.g., sender address, timestamp) to detect anomalies indicative of scams.
- **Libraries**:
  - **pandas**: Preprocesses metadata into structured formats (e.g., encoding sender domains).
  - **torch**: Constructs embeddings and trains a classification model for metadata features.
- **Why**: Metadata provides subtle signals (e.g., unusual domains or send times) that enhance detection when combined with text analysis.

3. Link Model (Hyperlink Analysis)

- **Role**: Classifies URLs as safe or unsafe by analyzing domain names, redirection patterns, or URL structures.
- **Libraries**:
  - **tldextract** or **regex**: Parses URLs into components (e.g., domain, subdomain).
  - **torch**: Trains a binary classifier for link safety.
- **Why**: Phishing often relies on deceptive links, and a dedicated model reduces false negatives by focusing on URL-specific patterns.

4. Combination Model

- **Role**: Aggregates probability scores from the Data Model, Metadata Model, and Link Model using a weighted ensemble or neural network layer.
- **Libraries**:
  - **torch**: Combines outputs into a final scam probability.
- **Why**: Integrating multiple signals improves robustness and accuracy, addressing complex scam patterns.

5. Confidence Score

- **Role**: Converts the model’s probability output (0–1) to a user-friendly confidence score (0–100).
- **Libraries**:

    - **numpy**: Scales probabilities to percentage values.
- **Why**: A percentage-based score aligns with user stories requiring clear, interpretable results for non-technical users.

6. LLM Explanation Layer

- **Role**: Uses a large language model to explain classification decisions by highlighting suspicious words or phrases.
- **Libraries**:
    - **transformers**: Leverages Hugging Face models for natural language explanations.
- **Why**: Explanations build user trust and meet requirements for transparency and simplicity.

7. Explanation Report

- **Role**: Generates structured reports listing suspicious tokens and classification rationale.
- **Libraries**:
    - **pandas**: Stores and formats report data.
    - **matplotlib** or **plotly**: Visualizes report data for dashboards.
- **Why**: Reports provide actionable insights for users and administrators, aiding model refinement.

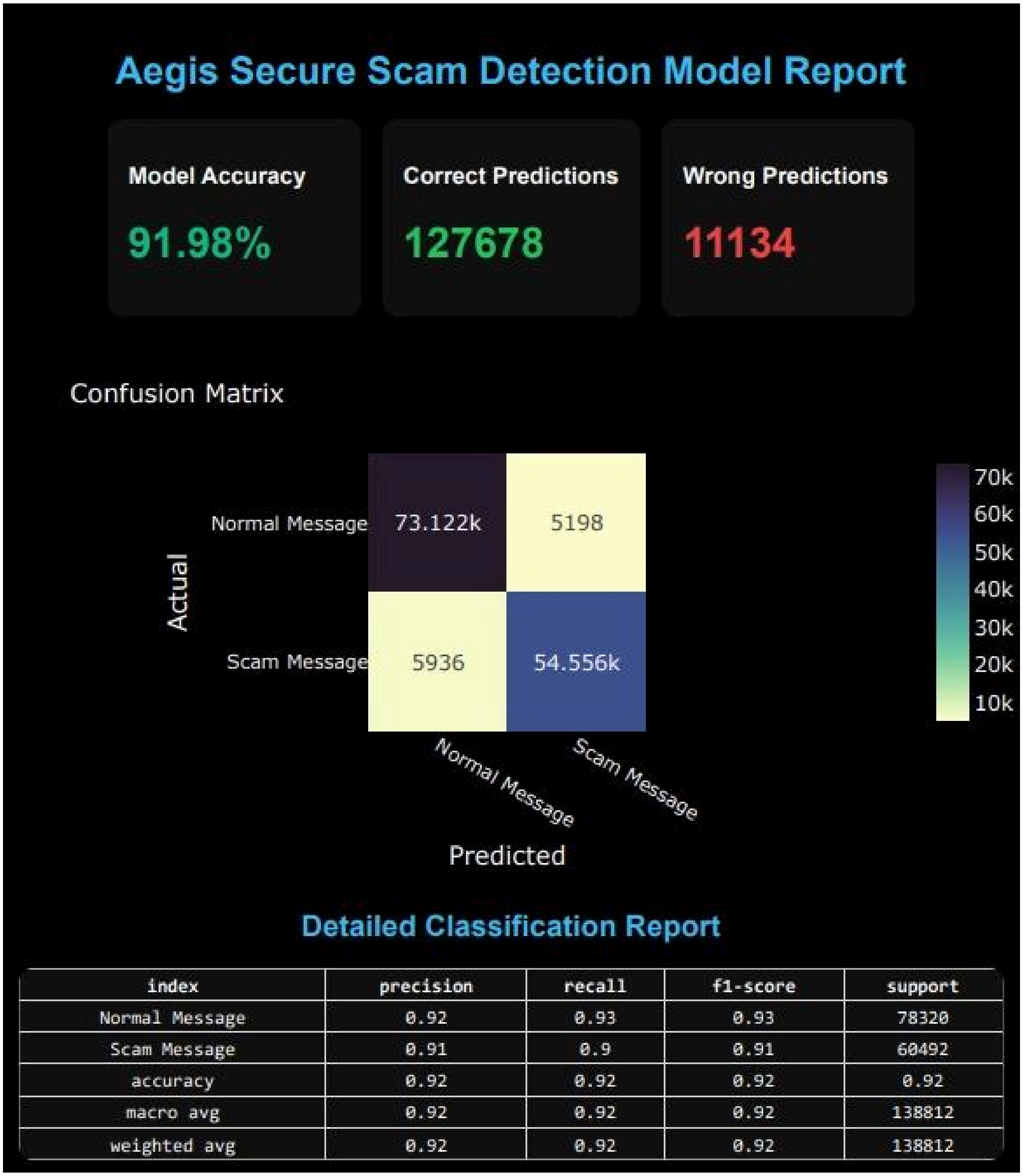
8. Analytics Dashboard

- **Role**: Displays historical scan results, including classification distributions and performance metrics (e.g., false positives/negatives).
- **Libraries**:
    - **streamlit** or **plotly-dash**: Builds interactive dashboards.
    - **matplotlib** or **seaborn**: Generates charts (e.g., pie charts for classification summaries).
- **Why**: Dashboards meet user requirements for tracking safety over time and monitoring system performance.

9. API Layer with FastAPI

- **Role**: Serves the trained models as REST APIs for cloud-based inference, supporting endpoints for scanning, explanations, and analytics.
- **Libraries**:
    - **fastapi**: Creates high-performance, asynchronous APIs.
    - **uvicorn**: Runs the FastAPI service.
- **Why**: FastAPI’s asynchronous capabilities and compatibility with PyTorch make it ideal for scalable cloud deployment, meeting requirements for real-time responses.
-

Model Classification and Accuracy Report



Aegis Secure Scam Detection Model Report

Model Accuracy

91.98%

Correct Predictions

127678

Wrong Predictions

11134

Confusion Matrix

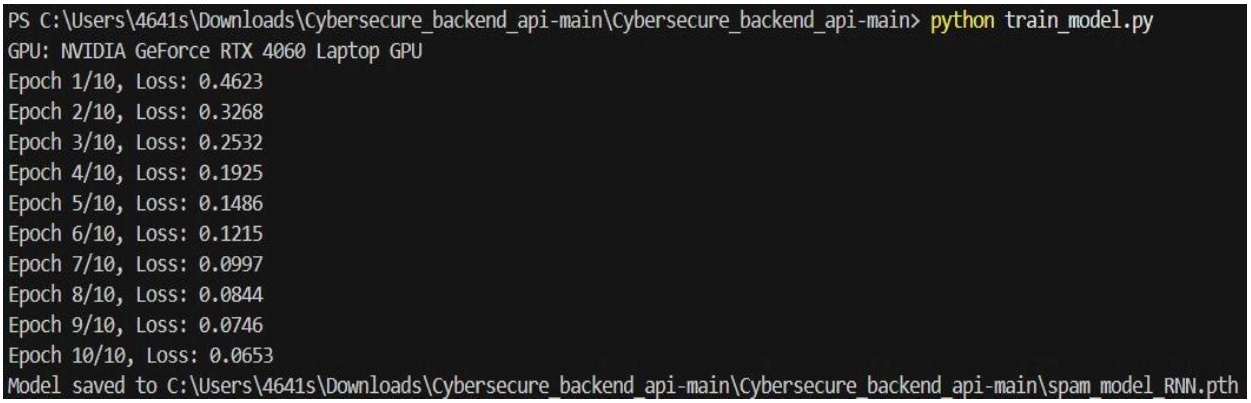
Actual \ Predicted	Normal Message	Scam Message
Normal Message	73.122k	5198
Scam Message	5936	54.556k

Detailed Classification Report

index	precision	recall	f1-score	support
Normal Message	0.92	0.93	0.93	78320
Scam Message	0.91	0.9	0.91	60492
accuracy	0.92	0.92	0.92	0.92
macro avg	0.92	0.92	0.92	138812
weighted avg	0.92	0.92	0.92	138812

Description: Summary of the current model’s performance, including metrics such as accuracy, *precision, recall, F1-score, and confusion matrix, based on evaluation results from* true.csv *and* pred.csv*. Will include a table or chart summarizing classification performance on the test set.*

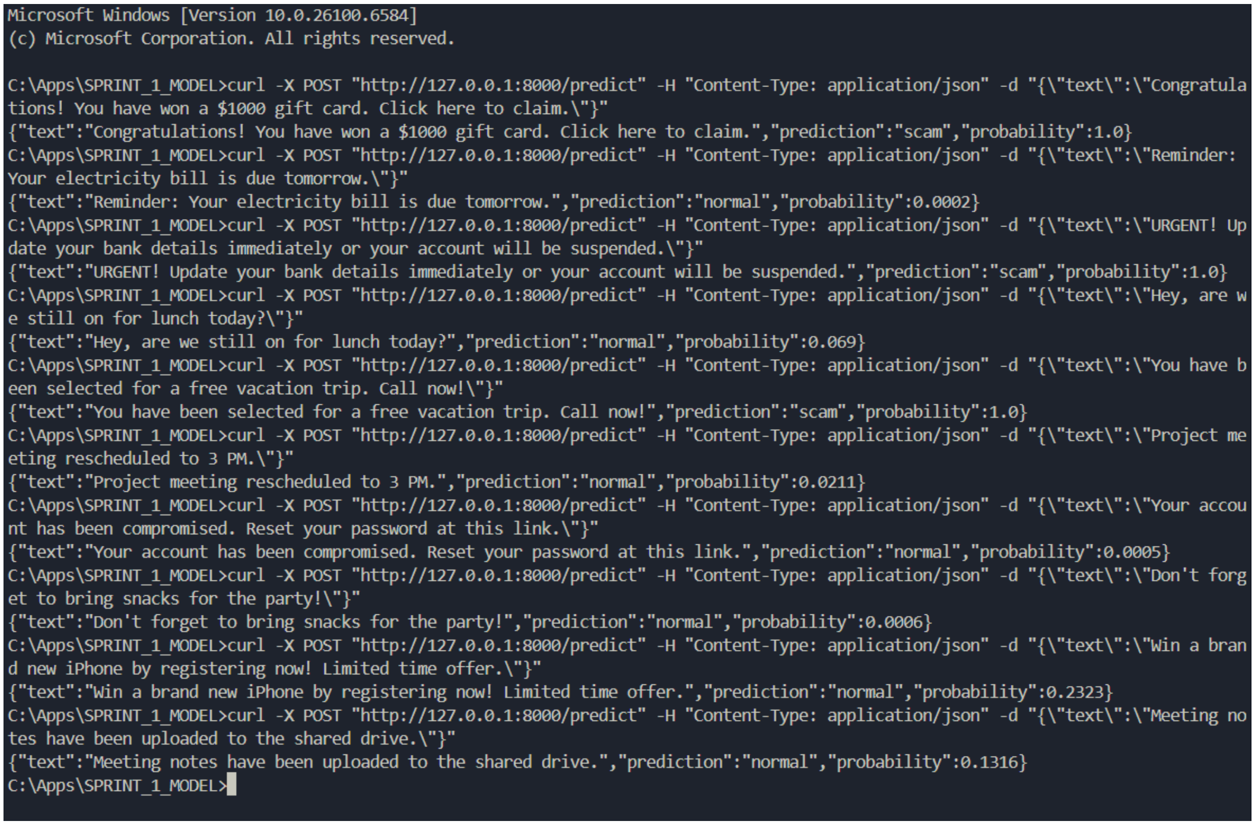
Training Screenshot



```
PS C:\Users\4641s\Downloads\Cybersecure_backend_api-main\Cybersecure_backend_api-main> python train_model.py
GPU: NVIDIA GeForce RTX 4060 Laptop GPU
Epoch 1/10, Loss: 0.4623
Epoch 2/10, Loss: 0.3268
Epoch 3/10, Loss: 0.2532
Epoch 4/10, Loss: 0.1925
Epoch 5/10, Loss: 0.1486
Epoch 6/10, Loss: 0.1215
Epoch 7/10, Loss: 0.0997
Epoch 8/10, Loss: 0.0844
Epoch 9/10, Loss: 0.0746
Epoch 10/10, Loss: 0.0653
Model saved to C:\Users\4641s\Downloads\Cybersecure_backend_api-main\Cybersecure_backend_api-main\spam_model_RNN.pth
```

Description: Screenshot of the training process output, showing epoch-wise loss values and training progress for the current message detection model.

Testing Screenshot



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Congratula
tions! You have won a $1000 gift card. Click here to claim.\"}"
{"text":"Congratulations! You have won a $1000 gift card. Click here to claim.","prediction":"scam","probability":1.0}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Reminder:
Your electricity bill is due tomorrow.\"}"
{"text":"Reminder: Your electricity bill is due tomorrow.","prediction":"normal","probability":0.0002}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"URGENT! Up
date your bank details immediately or your account will be suspended.\"}"
{"text":"URGENT! Update your bank details immediately or your account will be suspended.","prediction":"scam","probability":1.0}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Hey, are w
e still on for lunch today?\"}"
{"text":"Hey, are we still on for lunch today?","prediction":"normal","probability":0.069}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"You have b
een selected for a free vacation trip. Call now!\"}"
{"text":"You have been selected for a free vacation trip. Call now!","prediction":"scam","probability":1.0}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Project me
eting rescheduled to 3 PM.\"}"
{"text":"Project meeting rescheduled to 3 PM.","prediction":"normal","probability":0.0211}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Your accou
nt has been compromised. Reset your password at this link.\"}"
{"text":"Your account has been compromised. Reset your password at this link.","prediction":"normal","probability":0.0005}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Don't forg
et to bring snacks for the party!\"}"
{"text":"Don't forget to bring snacks for the party!","prediction":"normal","probability":0.0006}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Win a bran
d new iPhone by registering now! Limited time offer.\"}"
{"text":"Win a brand new iPhone by registering now! Limited time offer.","prediction":"normal","probability":0.2323}
C:\Apps\SPRINT_1_MODEL>curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d "{\"text\":\"Meeting no
tes have been uploaded to the shared drive.\"}"
{"text":"Meeting notes have been uploaded to the shared drive.","prediction":"normal","probability":0.1316}
C:\Apps\SPRINT_1_MODEL>
```

Role of Each Library in the Architecture

1. **os and dotenv**
   - **Purpose**: Loads environment variables (e.g., DATA_PATH, BATCH_SIZE, EPOCHS, LEARNING_RATE, MODEL_DIR, MAX_FEATURES) to configure the pipeline.
   - **How**: dotenv.load_dotenv() reads a .env file, and os.getenv() retrieves values, ensuring flexibility across environments.
   - **Why**: Separates configuration from code, enhancing security and portability for development and cloud deployment.
2. **pandas**
   - **Purpose**: Loads CSV data, preprocesses labels (mapping spam/scam → 1, ham → 0), removes missing values, and saves evaluation results.
   - **How**: Uses pd.read_csv(), df.apply(), df.dropna(), and pd.Series.to_csv() for data handling.
   - **Why**: Provides efficient data manipulation, critical for cleaning raw datasets and preparing evaluation outputs.
3. **scikit-learn**
   - **Purpose**: Performs TF-IDF vectorization and train-test splitting.
   - **How**: TfidfVectorizer(max_features=MAX_FEATURES) converts text to numerical features; train_test_split(test_size=0.4) splits data.
   - **Why**: TF-IDF is a proven method for text feature extraction, and scikit-learn's utilities are robust and widely adopted.
4. **torch and torch.nn**
   - **Purpose**: Builds and trains the ScamClassifier neural network with linear layers, ReLU activations, dropout, and sigmoid output.
   - **How**: Defines the model using nn.Sequential and moves it to GPU/CPU with .to(DEVICE).
   - **Why**: PyTorch's dynamic graph and ease of use support rapid prototyping in the current implementation, though TensorFlow will be adopted for mobile optimization.
5. **torch.optim**
   - **Purpose**: Implements the Adam optimizer for gradient-based weight updates.
   - **How**: Uses optim.Adam(model.parameters(), lr=LEARNING_RATE) to minimize BCELoss.
   - **Why**: Adam's adaptive learning rate accelerates convergence, suitable for deep learning tasks.
6. **numpy**

- ○ **Purpose**: Converts sparse TF-IDF matrices to dense arrays and processes evaluation outputs.
- ○ **How**: Used in ScamDataset for .toarray().astype('float32') and in evaluation for array operations.
- ○ **Why**: Provides efficient numerical computations, bridging scikit-learn and PyTorch.

7. **fastapi and uvicorn**
   - ○ **Purpose**: Planned for future implementation to serve the model via REST APIs.
   - ○ **How**: FastAPI will define endpoints (e.g., /predict, /explain) for inference and explanations; uvicorn will run the server.
   - ○ **Why**: FastAPI's asynchronous, high-performance framework is ideal for cloud deployment, ensuring real-time responses.
8. **matplotlib, seaborn, plotly (Planned for Future)**
   - ○ **Purpose**: Generate visualizations for the analytics dashboard and explanation reports.
   - ○ **How**: Will create pie charts, confusion matrices, and other plots to summarize classification results.
   - ○ **Why**: Visualizations meet user requirements for intuitive result presentation and historical tracking.
9. **transformers (Planned for Future)**
   - ○ **Purpose**: Powers the LLM Explanation Layer to highlight suspicious tokens and explain decisions.
   - ○ **How**: Will use Hugging Face models to generate natural language explanations.
   - ○ **Why**: Enhances user trust by providing clear, non-technical explanations, aligning with accessibility requirements.
10. **streamlit or plotly-dash (Planned for Future)**
    - ○ **Purpose**: Builds interactive dashboards for result visualization.
    - ○ **How**: Will display scan histories, classification distributions, and performance metrics.
    - ○ **Why**: Streamlit's simplicity and Plotly-Dash's interactivity support rapid dashboard development for user and admin interfaces.