ABSTRACT

Spam messages have become a common problem in everyday digital communication, especially in SMS and email platforms. This project presents a comprehensive and user-friendly SMS/Email Spam Classifier that goes beyond a basic machine learning model and transforms it into a practical, research-oriented web application. The main objective of this project is not only to classify messages as spam or ham, but also to make the classification process transparent, interactive, and adaptable.

The system is built using Python with Streamlit for the front-end interface and Scikit-learn for machine learning implementation. A Multinomial Naive Bayes algorithm combined with TF-IDF vectorization is used to perform text classification efficiently. To enhance text understanding, Natural Language Processing techniques such as tokenization, stopword removal, and stemming are implemented using NLTK. Additionally, multilingual support is integrated using language detection and automatic translation, allowing the system to process non-English messages as well.

Unlike traditional spam detection systems, this project focuses on interpretability. It explains why a message is classified as spam by highlighting the most influential words contributing to the decision. A debugging feature is also provided to visualize each preprocessing step, helping users understand how raw text is transformed before prediction. The system maintains prediction history, collects user feedback, and calculates real-world accuracy based on that feedback, making it more practical and realistic.

Furthermore, the application allows users to retrain the model by uploading custom datasets, enabling continuous improvement and adaptability. With features like theme customization, analytics dashboard, and confidence scoring, the project aims to combine machine learning performance with a smooth user experience.

Overall, this project demonstrates how a simple spam classifier can be extended into a full-fledged intelligent system that is interactive, explainable, and continuously improving. It shows that machine learning applications can be both technically strong and user-centered at the same time.

CONTENTS

1. INTRODUCTION

Today, Spam has become a major problem in communication over internet. It has been accounted that around 55% of all emails are reported as spam and the number has been growing steadily. Spam which is also known as unsolicited bulk email has led to the increasing use of email as email provides the perfect ways to send the unwanted advertisement or junk newsgroup posting at no cost for the sender. This chances has been extensively exploited by irresponsible organizations and resulting to clutter the mail boxes of millions of people all around the world. Spam has been a major concern given the offensive content of messages, spam is a waste of time. End user is at risk of deleting legitimate mail by mistake. Moreover, spam also impacted the economical which led some countries to adopt legislation. Text classification is used to determine the path of incoming mail/message either into inbox or straight to spam folder. It is the process of assigning categories to text according to its content. It is used to organized, structures and categorize text. It can be done either manually or automatically. Machine learning automatically classifies the text in a much faster way than manual technique. Machine learning uses pre-labelled text to learn the different associations between pieces of text and it output. It used feature extraction to transform each text to numerical representation in form of vector which represents the frequency of word in predefined dictionary. Text classification is important to structure the unstructured and messy nature of text such as documents and spam messages in a cost-effective way. Machine learning can make more accurate precisions in real-time and help to improve the manual slow process to much better and faster analysing big data. It is important especially to a company to analyse text data, help inform business decisions and even automate business processes.

In this project, machine learning techniques are used to detect the spam message of a mail. Machine learning is where computers can learn to do something 10 without the need to explicitly program them for the task. It uses data and produce a program to perform a task such as classification. Compared to knowledge engineering, machine learning techniques require

messages that have been successfully pre-classified. The pre-classified messages make the training dataset which will be used to fit the learning algorithm to the model in machine learning studio. A combination of algorithms are used to learn the classification rules from messages. These algorithms are used for classification of objects of different classes. These algorithms are provided with pre labelled data and an unknown text. After learning from the prelabelled data each of these algorithms predict which class the unknown text may belong to and the category predicted by majority is considered as final.

1.1 PROJECT DESCRIPTION

The SMS/Email Spam Classifier is a machine learning based web application developed to detect whether a given message is spam or legitimate (ham). In today's digital world, spam messages are increasing rapidly and can sometimes lead to fraud, phishing attacks, or privacy issues. Because of this, building an intelligent system that can automatically filter such messages becomes very important.

This project is designed not just as a basic spam detection model, but as a complete interactive application that allows users to understand, analyze, and even improve the model over time. The system is developed using Python and Streamlit for creating a simple and user-friendly web interface. The machine learning part uses the Multinomial Naive Bayes algorithm along with TFIDF vectorization, which is widely used for text classification problems.

The application follows a structured machine learning pipeline. First, the input message goes through preprocessing steps such as converting text to lowercase, tokenization, removing punctuation and special characters, eliminating stopwords, and stemming words to their root form. These steps help in reducing noise and improving prediction accuracy. After preprocessing, the text is converted into numerical format using TF-IDF and then passed to the trained classifier to determine whether it is spam or not.

One of the key highlights of this project is its explainability feature. Instead of just giving a prediction result, the system also shows the confidence score and the most important words that influenced the classification decision. This makes the model more transparent and easier to understand. A debug mode is also included where users can see step-by-step text transformation during preprocessing, which is especially useful for learning purposes.

Additionally, the project includes a prediction history module that stores previous results and allows users to provide feedback on whether the prediction was correct or incorrect. Based on this feedback, a real-world accuracy metric is calculated. The system also supports retraining by allowing users to upload new datasets, making it flexible and adaptable.

Overall, this project combines machine learning, natural language processing, and interactive web design to create a practical spam detection system. It not only performs classification but also focuses on usability, transparency, and continuous improvement, which makes it more realistic and applicable in real-world scenarios.

The SMS/Email Spam Classifier is an intelligent web-based application developed to automatically identify whether a given message is spam or legitimate (ham). With the rapid growth of digital communication, spam messages have become a serious issue. These unwanted messages not only disturb users but can also lead to phishing attacks, scams, and data theft. Because of this, an automated and reliable spam detection system is very important in today's environment.

This project started as a basic machine learning model but was later improved and transformed into a complete, feature-rich web application. The goal was not only to build a classifier but also to make it interactive, explainable, and adaptable for real-world usage. The system is built using Python programming language. Streamlit is used to create the frontend interface, which makes the application simple, clean, and easy to use even for non-technical users.

For the machine learning part, the project uses the Multinomial Naive Bayes algorithm, which is well-suited for text classification problems. The textual data is converted into numerical form using TF-IDF (Term Frequency–Inverse Document Frequency) vectorization. TF-IDF helps in giving more importance to unique and meaningful words while reducing the weight of common words.
This improves the accuracy of classification.

Before classification, the message goes through several preprocessing steps. These include:

- Converting text into lowercase to maintain uniformity
- Tokenizing the sentence into individual words
- Removing punctuation and special characters

- Eliminating stopwords such as “is”, “the”, “at”, etc.
- Applying stemming to reduce words to their root form (for example, “winning” becomes “win”)

These preprocessing steps are necessary because raw text usually contains noise, and cleaning it improves model performance.

One of the major highlights of this project is its explainability feature. Most spam classifiers only show the result as “Spam” or “Not Spam”, but this system also provides a confidence score and displays the top words that influenced the prediction. This helps users understand *why* the message was classified in a certain way. There is also a debug mode where users can see each preprocessing stage step-by-step, which makes the system educational as well.

Another important feature is multilingual support. The system can detect the language of the input message and translate non-English messages into English before classification. This increases usability and makes the model more flexible.

The project also includes a history and analytics module. Every prediction is stored, and users can give feedback indicating whether the prediction was correct or incorrect. Based on this feedback, a custom real-world accuracy score is calculated. This feature makes the system more practical, because real-world performance can be different from training accuracy.

Additionally, users can retrain the model by uploading their own dataset through the interface. The system automatically validates and cleans the dataset before retraining. This makes the model adaptable and capable of continuous improvement over time.

From a design perspective, the application includes features like dark/light theme toggle, sidebar navigation, organized layout, and graphical visualizations. These improvements make the user experience better and more professional.

1.1.1 OBJECTIVES

The main objective of this project is to develop an intelligent and user-friendly SMS/Email Spam Classifier that can accurately detect whether a message is spam or legitimate (ham). The system is designed not only to perform classification but also to make the process understandable and interactive for users.

The specific objectives of this project are as follows:

1. **To build an effective spam detection model**

   Develop a machine learning model using Multinomial Naive Bayes algorithm that can classify messages with good accuracy and efficiency.

2. **To implement text preprocessing techniques**

   Apply Natural Language Processing (NLP) methods such as tokenization, stopword removal, punctuation cleaning, and stemming to improve the quality of text data before classification.

3. **To use TF-IDF vectorization for feature extraction**

   Convert textual data into numerical format using TF-IDF so that important and meaningful words get higher importance during classification.

4. **To design an interactive web application**

   Create a simple and user-friendly interface using Streamlit where users can easily input messages and view results.

5. **To provide prediction confidence and interpretability**

   Display confidence scores and highlight the most influential words responsible for the prediction, so users can understand why a message is classified as spam or not.

6. **To include multilingual support**

   Detect different languages and translate non-English messages into English before processing, making the system more flexible and practical.

7. **To implement a feedback mechanism**

   Allow users to mark predictions as correct or incorrect and calculate real-world accuracy based on user feedback.

8. **To enable model retraining**

   Provide an option for users to upload their own datasets and retrain the model, allowing continuous improvement of the system.

9. **To maintain prediction history and analytics**

   Store previous predictions and generate simple analytics to monitor performance and usage patterns.

Overall, the objective of this project is not just to create a spam classifier, but to build a complete intelligent system that is accurate, transparent, user-centered, and capable of improving over time.

1.1.2 SYSTEM OVERVIEW

The SMS/Email Spam Classifier is a web-based machine learning application developed to automatically detect whether a given message is spam or legitimate (ham). The system combines Natural Language Processing (NLP), machine learning algorithms, and an interactive user interface to provide accurate and understandable predictions.

At a high level, the system consists of three main components: the user interface, the processing and machine learning module, and the data storage module. All these components work together to provide a smooth and efficient spam detection experience.

When a user enters a message into the application, the system first performs language detection. If the message is not in English, it is automatically translated into English to ensure consistent processing. After this, the message goes through a preprocessing stage. In this stage, the text is converted to lowercase, tokenized into individual words, cleaned by removing punctuation and special characters, filtered to remove stopwords, and finally stemmed to reduce words to their root form. This step is important because raw text data usually contains noise and inconsistencies.

Once preprocessing is completed, the cleaned text is transformed into numerical format using TFIDF vectorization. This technique assigns weight to words based on their importance in the message compared to the entire dataset. The numerical representation is then passed to the trained Multinomial Naive Bayes classifier, which predicts whether the message is spam or ham.

The system does not only provide a simple output. It also displays a confidence score indicating how certain the model is about its prediction. Additionally, it shows the most influential words that contributed to the classification result. This makes the system more transparent and helps users understand the reasoning behind the prediction.

Another important part of the system is the history and feedback module. Every prediction is stored in a local history file, and users can provide feedback by marking predictions as correct or incorrect. Based on this feedback, the system calculates a custom accuracy score, which reflects real-world performance rather than just training accuracy.

The system also includes a retraining feature where users can upload their own dataset in CSV format. The application validates and processes the dataset before retraining the model. This allows the system to adapt and improve over time.

1.1.3 MODULES AND FUNCTIONALITIES

The SMS/Email Spam Classifier system is divided into different modules, where each module performs a specific task. This modular structure makes the system organized, easy to maintain, and scalable for future improvements. The main modules and their functionalities are described below:

1. User Interface Module (Frontend)

This module is developed using Streamlit and is responsible for user interaction. It provides a clean and simple interface where users can enter messages and view prediction results.

Functionalities:

- Text input area for entering SMS or email messages
- Display of prediction result (Spam or Not Spam)
- Showing confidence score of the prediction
- Theme toggle option (Dark/Light mode)
- Sidebar navigation for switching between sections
- Expandable sections for viewing detailed processing steps

This module ensures that even non-technical users can easily use the system without confusion.

2. Text Preprocessing Module

This module is implemented in the utils.py file and handles all Natural Language Processing (NLP) tasks before classification. It prepares raw text data for machine learning.

Functionalities:

- Converting text to lowercase
- Tokenizing sentences into individual words
- Removing punctuation and special characters
- Removing stopwords (common words with less meaning)
- Applying stemming to reduce words to root form
- Debug mode to display step-by-step preprocessing stages

This module plays a very important role because proper text cleaning directly affects model accuracy.

3. Language Detection and Translation Module

This module enhances the system by supporting multilingual input.

Functionalities:

- Detecting the language of the input message
- Automatically translating non-English messages into English
- Ensuring consistent processing for all messages

This feature makes the system more flexible and usable in real-world situations where messages may come in different languages.

4. Feature Extraction Module

This module converts cleaned text into numerical format so that the machine learning model can process it.

Functionalities:

- Applying TF-IDF vectorization
- Assigning weights to important words
- Transforming text data into numerical vectors

TF-IDF helps in giving more importance to meaningful and unique words, which improves prediction performance.

5. Classification Module

This is the core machine learning module of the system.

Functionalities:

- Using the Multinomial Naive Bayes algorithm for classification
- Predicting whether a message is Spam or Ham
- Calculating probability scores
- Displaying confidence percentage

This module makes the final decision based on trained data and extracted features.

6. Model Interpretability Module

Unlike many basic spam classifiers, this project includes an explanation feature.

Functionalities:

- Identifying top words influencing the prediction
- Displaying them in graphical format (bar chart)
- Helping users understand why a message was marked as spam This improves transparency and makes the system more trustworthy.

7. History and Analytics Module

This module stores and analyzes previous predictions.

Functionalities:

- Saving prediction history in a JSON file
- Recording user feedback (Correct/Incorrect)
- Calculating real-world accuracy based on feedback
- Displaying basic analytics and charts

This module makes the system more practical and helps in evaluating performance over time.

8. Model Training and Retraining Module

This module is implemented in train_model.py and is responsible for training and updating the model.

Functionalities:

- Loading and cleaning the dataset
- Validating column formats (v1/v2 or target/text)
- Training the Naive Bayes model
- Saving trained model and vectorizer using pickle
- Allowing users to upload custom datasets for retraining

This feature ensures that the model can adapt and improve when new data is provided.

1.1.4 TECHNOLOGY STACK

Got it You want the **Technology Stack** section written in the same format/style as the example you shared.

Here is your **Technology Stack** for the SMS/Email Spam Classifier project written in that format and in a natural human style

Technology Stack

The following technologies are used in the SMS/Email Spam Classifier project:

• **Frontend:** Streamlit (for building the interactive web application interface)

• **Backend:** Python

• **Machine Learning & NLP:**

Python, Scikit-learn (Multinomial Naive Bayes), TF-IDF Vectorization, NLTK (Natural Language Toolkit), langdetect, deep-translator

• **Data Handling & Processing:**

Pandas, NumPy

• **Data Visualization:**

Matplotlib

• **Data Storage & Persistence:**

Pickle (for saving trained model and vectorizer), JSON (for storing prediction history)

• **Dataset:**

spam.csv dataset (5,574 SMS messages)

• **Tools & IDEs:**

Jupyter Notebook, VS Code, PyCharm

• **Operating Systems:**

Windows, Linux

1.1.5 METHODOLOGY

The methodology of the SMS/Email Spam Classifier project follows a structured machine learning approach, starting from data collection to model deployment in a web application. The main aim was to design a system that is not only accurate but also easy to use and understand.

1. Data Collection

The project uses a publicly available dataset (spam.csv) which contains 5,574 SMS messages labeled as either "spam" or "ham." The dataset was first examined to understand its structure, column names, and class distribution. Some minor cleaning was required to remove unnecessary columns and handle formatting issues.

2. Data Preprocessing

Raw text data cannot be directly used for machine learning, so preprocessing is an important step. The following techniques were applied:

- **Lowercasing:** All text was converted to lowercase to maintain uniformity.
- **Tokenization:** Sentences were split into individual words.
- **Removal of Punctuation and Special Characters:** To reduce noise in the data.

- **Stopword Removal:** Common words such as “is”, “the”, “at” were removed because they do not contribute much to classification.
- **Stemming:** Words were reduced to their root form (for example, “winning” becomes “win”).

These steps helped in cleaning the text and improving the quality of features used for training.

3. Feature Extraction

After preprocessing, the text data was converted into numerical format using **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorization. This technique assigns higher weights to important and unique words while reducing the importance of common words. This step is necessary because machine learning models cannot directly understand textual data.

4. Model Selection and Training

For classification, the **Multinomial Naive Bayes** algorithm was chosen. This algorithm is widely used for text classification problems because it is simple, fast, and performs well with word frequency features like TF-IDF.

The dataset was divided into training and testing sets. The model was trained using the training data and evaluated on the testing data to measure its performance. Accuracy and basic evaluation metrics were analyzed to ensure the model was performing satisfactorily.

5. Model Saving

Once the model achieved acceptable performance, it was saved using the pickle library. The trained TF-IDF vectorizer was also saved separately. This allows the model to be reused without retraining every time the application runs.

6. Web Application Development

The trained model was integrated into a Streamlit web application. The application allows users to:

- Enter a message for classification
- View prediction results with confidence score
- See the most influential words affecting the decision
- View step-by-step preprocessing in debug mode

This step transforms the machine learning model into a practical and interactive tool.

7. Feedback and Retraining Mechanism

To make the system more realistic, a feedback feature was implemented. Users can mark predictions as correct or incorrect. Based on this feedback, a custom real-world accuracy is calculated. Additionally, users can upload new datasets to retrain the model, allowing continuous improvement.

1.1.6 EXPECTED OUTCOMES

The expected outcome of the SMS/Email Spam Classifier project is to develop a reliable and userfriendly system that can accurately classify messages as spam or legitimate (ham). By implementing machine learning and Natural Language Processing techniques, the system should be able to identify unwanted or suspicious messages with good accuracy and efficiency.

One of the primary expected results is achieving high classification accuracy using the Multinomial Naive Bayes algorithm with TF-IDF vectorization. The system should be able to correctly detect common spam patterns such as promotional offers, suspicious links, and fraudulent messages while minimizing false predictions.

Another expected outcome is improved user understanding of how spam detection works. Through features like confidence scoring, top influential words, and step-by-step preprocessing visualization, users should be able to see why a particular message was classified as spam. This makes the system more transparent and educational, not just functional.

The project is also expected to provide multilingual support, meaning it should successfully detect and translate non-English messages before classification. This increases the practical usability of the system in real-world scenarios.

Additionally, the implementation of a feedback mechanism should help in tracking real-world performance. Over time, the system's custom accuracy score based on user feedback should reflect how well the model performs outside the training dataset. The retraining feature is also expected to allow continuous improvement when new datasets are provided.

From a usability perspective, the final application should have a clean and responsive interface with easy navigation, theme switching options, and proper visualization of analytics. The system should run smoothly on standard operating systems like Windows and Linux without requiring high-end hardware.

Overall, the expected outcome is not just a working spam classifier, but a complete intelligent application that is accurate, explainable, adaptable, and practical for real-world use. It should demonstrate how machine learning concepts can be applied to solve everyday communication problems in an effective way.

1.2 COMPANY PROFILE

Target Users:

- Individuals who want to filter unwanted messages and protect personal communication
- Organizations aiming to monitor spam in email and SMS communication
- Students, researchers, and developers interested in machine learning applications for text classification

Products and Services: Although this project is primarily a software application, its functionalities mimic real company offerings in:

- **Spam Detection:** Accurate identification of spam and ham messages
- **Multilingual Support:** Translation of non-English messages for consistent processing
- **Analytics & Reporting:** Tracking predictions, feedback, and accuracy metrics
- **Custom Model Training:** Retraining the system using new datasets to improve performance

Technologies Used: The project uses modern technologies and tools such as Python, Streamlit, Scikit-learn, NLTK, TF-IDF Vectorization, JSON, and Pickle for model persistence, making it similar to enterprise-level AI solutions.

Vision and Future Scope: The project aims to provide a platform that demonstrates practical AI application in communication security. With continuous retraining, feedback-driven accuracy, and

possible integration into email/SMS clients, this system can evolve into a professional-grade spam detection solution.

2. LITERATURE SURVEY

2.1 EXISTING AND PROPOSED SYSTEM

Existing System

In the existing system, spam detection is usually performed using basic filters or simple machine learning models. Traditional spam filters rely on keyword matching, blacklists, or manually defined rules to identify spam messages. While these methods can catch obvious spam, they have several limitations:

- They often fail to detect new or cleverly disguised spam messages.
- There is no support for multiple languages, so non-English messages are not accurately classified.
- Most existing systems provide only a binary output (spam or not spam) without explaining why a message was classified a certain way.
- Users cannot provide feedback to improve the system or track real-world accuracy.
- Retraining the model with new data is often difficult or not supported, making it less adaptable.

Overall, the existing systems are functional but lack intelligence, transparency, and adaptability, which reduces their effectiveness in dynamic real-world scenarios.

Proposed System

The proposed SMS/Email Spam Classifier overcomes the limitations of existing systems by using machine learning, NLP, and interactive web technologies. The key improvements include:

- Machine Learning-Based Detection: Uses the Multinomial Naive Bayes algorithm with TF-IDF vectorization for accurate text classification.

- Explainability: Shows confidence scores and highlights the top words influencing the spam prediction, helping users understand *why* a message was classified as spam.
- Multilingual Support: Detects the language of the input message and translates nonEnglish messages into English before processing.
- Feedback and Real-World Accuracy: Users can provide feedback on predictions, allowing the system to calculate accuracy based on actual use.
- Retraining Capability: Users can upload new datasets to retrain the model, enabling continuous improvement.
- Interactive Web Interface: Provides a user-friendly and responsive design using Streamlit, with theme switching, step-by-step debugging, and visualization of processing steps.

In short, the proposed system is more intelligent, adaptable, and user-centered, providing a practical solution for real-world spam detection that is both functional and educational.

2.2 FEASIBILITY STUDY

Before developing the SMS/Email Spam Classifier, a feasibility study was conducted to evaluate whether the project is practical, cost-effective, and technically achievable. The feasibility study considers technical, economic, operational, and social aspects.

1. Technical Feasibility

The project is technically feasible because it uses widely available and well-supported technologies: Python for backend and machine learning, Streamlit for frontend, and libraries like Scikit-learn, NLTK, and TF-IDF for Natural Language Processing. The system does not require high-end hardware and can run on standard computers with Windows or Linux. Pretrained models and open-source datasets are used, reducing the technical complexity.

2. Economic Feasibility

The project is economically feasible since it relies primarily on open-source software and free development tools. Python, Streamlit, Scikit-learn, and other required libraries are free to use. There is no need for expensive hardware or paid datasets. The main cost is developer time and effort.

3. Operational Feasibility

The system is designed to be user-friendly with a simple web interface. Users do not need technical expertise to classify messages, view results, or provide feedback. The feedback and retraining features make it operationally practical because it can continuously improve based on real-world use.

4. Schedule/Time Feasibility

The project development timeline is realistic. Core components like data preprocessing, model training, and web integration can be completed in a reasonable timeframe. Testing, debugging, and deployment can be done incrementally, allowing smooth project completion.

5. Social Feasibility

Spam messages are a real-world problem affecting millions of users. Developing an accurate and explainable spam detection system has social relevance as it helps users protect themselves from scams and phishing. The system also provides educational value for students or researchers learning about NLP and machine learning.

2.3 TOOLS AND TECHNOLOGIES USED

The SMS/Email Spam Classifier project is built using a combination of modern programming languages, libraries, and tools to ensure efficiency, accuracy, and user-friendliness. The backend and machine learning components are developed in Python, which provides a rich ecosystem for data processing and model building. For the frontend, Streamlit is used to create an interactive and responsive web interface that allows users to input messages, view predictions, and explore processing steps. The machine learning model is implemented using Scikit-learn, specifically the Multinomial Naive Bayes algorithm, while TF-IDF vectorization is employed for feature extraction. Natural Language Processing is handled using NLTK, which manages tokenization, stopword removal, and stemming. To support multilingual input, the system uses langdetect for language detection and deep-translator for translating non-English messages into English.

For data handling and preprocessing, Pandas and NumPy are used to manipulate datasets and perform numerical operations. The trained models and vectorizers are saved using Pickle, while

prediction history and user feedback are stored in JSON format. Matplotlib is used for visualizing influential words and analytics charts, providing insights into the model's decisions. Development and experimentation are carried out using Jupyter Notebook, VS Code, and PyCharm, which streamline coding, testing, and debugging. The project uses the publicly available spam.csv dataset containing 5,574 labeled messages and is compatible with both Windows and Linux operating systems. Overall, the combination of these tools and technologies enables the system to be accurate, interactive, and adaptable, making it suitable for real-world spam detection scenarios.

2.4 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements

The SMS/Email Spam Classifier project does not require high-end hardware and can run on standard computing systems. The hardware requirements are as follows:

- **Processor:** Minimum dual-core processor; recommended quad-core processor for faster performance
- **RAM:** Minimum 4 GB; recommended 8 GB for retraining with large datasets
- **Storage:** Minimum 500 MB of free space for Python, libraries, and project files; recommended 1 GB
- **Input/Output Devices:** Keyboard, mouse, and monitor for user interaction
- **Internet Connection:** Required for language translation and downloading libraries

Software Requirements

The software requirements focus on the programming environment, libraries, and development tools necessary for implementing and running the project:

- **Operating System:** Windows or Linux
- **Programming Language:** Python 3.x
- **Frontend Framework:** Streamlit for the web interface
- **Machine Learning Libraries:** Scikit-learn (Naive Bayes), TF-IDF Vectorizer

- **NLP Libraries:** NLTK (tokenization, stemming, stopword removal), langdetect, deeptranslator
- **Data Handling Libraries:** Pandas, NumPy
- **Data Visualization:** Matplotlib
- **Model Persistence:** Pickle (for saving models), JSON (for storing history and feedback)
- **Development Tools / IDEs:** Jupyter Notebook, VS Code, PyCharm
- **Web Browser:** Modern browser to run Streamlit application

- **Optional Tools:** Git for version control

3. SOFTWARE REQUIREMENTS SPECIFICATION

The Software Requirements Specification (SRS) describes the overall description, scope, users, and requirements of the **SMS/Email Spam Classifier** system. This document explains what the system does, the technologies used, and the features required for proper functioning of the application. The system is designed as a web-based application with an AI-powered backend capable of classifying messages in real-time, analyzing NLP preprocessing steps, and maintaining a prediction history.

1. Programming Language

- **Python 3.x:** Core programming language used for developing the machine learning model, backend logic, NLP preprocessing, and web application.

2. Machine Learning and AI Libraries

- **Scikit-learn:** For building and training the Multinomial Naive Bayes model and evaluating model performance.
- **NLTK:** For tokenization, stopword removal, and stemming in NLP preprocessing.
- **TF-IDF Vectorizer (from Scikit-learn):** For converting textual data into numerical features.
- **langdetect:** For detecting the language of input messages.
- **deep-translator:** For translating non-English messages into English before classification.
- **pickle:** For saving and loading trained models and vectorizers.

3. Web Development Frameworks

- **Streamlit:** For creating the interactive web-based interface, integrating backend logic, and displaying results.
- **JSON:** For storing message history and user feedback logs.

4. Frontend Technologies

- **Streamlit Components:** For building the user interface, including sidebars, theme toggle, containers, and charts.
- **Plotting Libraries:** Such as **Matplotlib** or **Plotly** for visualizing analytics and model interpretability results.

5. Development Tools

- **IDE:** VS Code, PyCharm, or Jupyter Notebook for coding, model training, debugging, and testing.
- **Version Control:** Git and GitHub for source code management and collaboration.

6. Operating System

- Windows, Linux, or MacOS for development and deployment.
- Can also run on cloud platforms for remote access and sharing.

7. Browser Requirements

- Modern web browsers such as **Google Chrome**, **Mozilla Firefox**, or **Microsoft Edge** are required to access the web-based dashboard.

8. Key Features & Functionalities

1. **Message Classification:**
    - Classifies SMS/Email messages as **Spam** or **Ham**.
    - Provides a **confidence score** for each prediction.
2. **NLP Pipeline Visualization:**
    - Displays step-by-step processing, including lowercasing, tokenization, stopword removal, and stemming.
3. **Multilingual Support:**
    - Detects language of the message and translates non-English messages into English before classification.
4. **History Tracking & Analytics:**
    - Stores all predictions and feedback.
    - Shows usage patterns, spam vs ham charts, and real-world accuracy metrics.
5. **Model Retraining:**
    - Allows users to upload custom CSV datasets to retrain the model.
    - Handles different column formats and automatically cleans the data.
6. **User Feedback Loop:**
    - Users can mark predictions as correct or incorrect, improving tracking of real-world accuracy.

7. **UI/UX Features:**
    - Theme toggle (Dark/Light mode).
    - Responsive layout with sidebars, charts, and expandable sections.

3.2 SCOPE AND OBJECTIVE

The SMS/Email Spam Classifier is a **web-based application** designed to help users identify spam messages efficiently. It supports multiple languages, translates non-English messages, and provides real-time predictions with confidence scores. The system also keeps a **history of predictions**, allows user feedback, and enables retraining with custom datasets.

The application is intended for **general users, researchers, and students** who want a reliable spam detection tool and a way to learn about **NLP and machine learning pipelines** in an interactive manner. The interface is **responsive and user-friendly**, with features like theme toggle, step-by-step processing display, and visual analytics.

Objective

The main objectives of the project are:

1. **Accurate Spam Detection:** Classify messages as spam or ham using **Multinomial Naive Bayes** and TF-IDF vectorization.
2. **Transparency:** Show step-by-step NLP processing (lowercasing, tokenization, stopword removal, stemming) to explain predictions.
3. **Multilingual Support:** Detect message language and translate non-English messages for accurate classification.
4. **Feedback and Analytics:** Collect user feedback to calculate real-world accuracy and display insights in charts.
5. **Retraining Capability:** Allow users to upload datasets and retrain the model to improve performance.
6. **User-Friendly Interface:** Provide an interactive, clean, and responsive web interface with theme toggle and analytics dashboard.

3.3 PROBLEM STATEMENT

With the rapid growth of digital communication, people receive hundreds of SMS and email messages every day. Unfortunately, a significant portion of these messages are spam, which can range from annoying advertisements to malicious phishing attempts. Manually identifying and filtering spam is time-consuming, error-prone, and often ineffective, especially for non-technical users.

Existing spam filters either rely on basic keyword matching, which is inaccurate, or complex systems that lack transparency, making it difficult for users to understand why a message was classified as spam. Moreover, many systems do not support multilingual messages, provide confidence scores, or allow users to give feedback to improve accuracy.

This project aims to solve these issues by developing a smart, interactive, and educational SMS/Email Spam Classifier. The system will accurately detect spam messages using machine learning and NLP techniques, support multiple languages, show the step-by-step message processing, store prediction history, and allow users to retrain the model with new data.

In essence, the problem is the lack of an easy-to-use, accurate, and transparent spam detection system that also allows users to learn from the classification process and improve the model over time.

Functional Requirements

The functional requirements define what the SMS/Email Spam Classifier system should do to meet its objectives:

- The system must allow users to input SMS or email messages for classification.
- It should classify messages as **Spam** or **Ham** using the trained machine learning model.
- The system must provide a **confidence score** for each prediction.
- It should display **step-by-step NLP preprocessing** for each message, including lowercasing, tokenization, stopword removal, and stemming.
- The system must detect the **language of messages** and translate non-English texts into English before classification.
- Users should be able to **view a history log** of all classified messages along with timestamps, labels, confidence scores, and user feedback.

Non-Functional Requirements

The non-functional requirements specify how the system performs and behaves under different conditions:

- **Performance:** The system should classify messages in near real-time with minimal delay.
- **Reliability:** The classifier must provide consistent predictions across different message types and languages.
- **Usability:** The web interface should be **easy to navigate**, interactive, and accessible from desktops or laptops.
- **Scalability:** The system should handle multiple messages, feedback entries, and retraining operations without significant slowdown.
- **Portability:** The application must run on **Windows, Linux, or MacOS**, and can be deployed on cloud platforms.
- **Security:** User feedback, history logs, and uploaded datasets must be securely stored to prevent unauthorized access or data loss.
- **Maintainability:** The modular design should allow easy updates to the **machine learning model**, NLP preprocessing, or frontend interface.

4. SYSTEM DESIGN

This SMS/Email Spam Classifier project is designed as a complete, end-to-end web application that goes far beyond a simple machine learning script. It is built using a modern Python-based architecture where Streamlit powers the interactive frontend and connects seamlessly with the backend logic written in Python. The application handles text preprocessing through a utility module and uses a dedicated training script to build and retrain the machine learning model. For classification, it relies on TF-IDF vectorization combined with a Multinomial Naive Bayes algorithm, which is well-known for performing efficiently in text classification tasks. Natural Language Processing techniques such as tokenization, stopword removal, and stemming are implemented using NLTK to ensure that the text data is cleaned and standardized before prediction. The trained model and vectorizer are stored using pickle, while user prediction history is maintained in JSON format for persistence.

The system works through a well-structured pipeline that begins with preprocessing the input message. The text is first converted into lowercase to maintain consistency, then tokenized into individual words. Unnecessary characters and punctuation are removed, followed by stopword elimination to filter out common but meaningless words. After that, stemming reduces words to their base form, which helps improve model performance. Once preprocessing is complete, TFIDF transforms the cleaned text into numerical features by assigning importance scores to words based

on their frequency and uniqueness. These features are then passed into the Naive Bayes classifier, which predicts whether the message is spam or not spam and also provides a confidence score. One of the most interesting aspects of the system is that it explains its decision by highlighting the top words that influenced the classification, making the model more transparent and easier to understand.

In addition to basic classification, the application includes multilingual support by detecting the input language and translating non-English messages into English before processing. It also provides a testing phase where users can view each preprocessing step, which makes the system educational as well as functional. Every prediction is stored in a history log, and users can give feedback on whether the prediction was correct or incorrect. This feedback is used to calculate a custom real-world accuracy metric, making the system feel more dynamic and practical. Moreover, users can retrain the model directly from the interface by uploading their own dataset, which allows the application to adapt to new types of spam messages. Overall, this project is not just a classifier but a complete, user-friendly, and research-oriented system that combines machine learning, NLP, data persistence, and interactive design in a single platform.

This SMS and Email Spam Classifier is designed as a complete system rather than just a basic machine learning model running in the background. The architecture follows a clean separation between the frontend and backend components. The frontend is developed using Streamlit, which provides an interactive and responsive web interface where users can input messages, view predictions, switch themes, and explore analytics. The backend logic is organized into separate Python modules to maintain clarity and modularity. The preprocessing logic is handled inside a utility file, while the model training and retraining pipeline is managed through a dedicated training script. This separation makes the system easier to maintain, update, and scale in the future, even though at first it started as a simple experimental script.

At the core of the system lies the machine learning pipeline. When a user enters a message, it does not go directly to the classifier. Instead, it passes through several preprocessing stages to ensure the input is standardized. The text is first converted to lowercase to remove case sensitivity issues. Then tokenization breaks the sentence into individual words so that each word can be analyzed separately. Special characters, punctuation, and unnecessary symbols are removed because they usually do not contribute to meaningful classification. Stopwords such as “is”, “the”, and “at” are filtered out since they appear frequently in both spam and non-spam messages and do not help the model differentiate effectively. After that, stemming reduces words to their root forms, for example

“winning”, “winner”, and “wins” may all reduce to a common base form. This step slightly reduces vocabulary size and improves efficiency, although sometimes it makes the words look a bit unnatural.

Once preprocessing is completed, the cleaned text is converted into numerical features using TFIDF vectorization. This technique assigns weights to words based on how important they are in a particular message compared to the entire dataset. Words that appear frequently in spam messages but rarely in normal messages get higher importance scores. These numerical vectors are then passed into a Multinomial Naive Bayes classifier. This algorithm is chosen because it performs very well on text-based classification problems and is computationally efficient.

4.1 ARCHITECTURE DIAGRAM

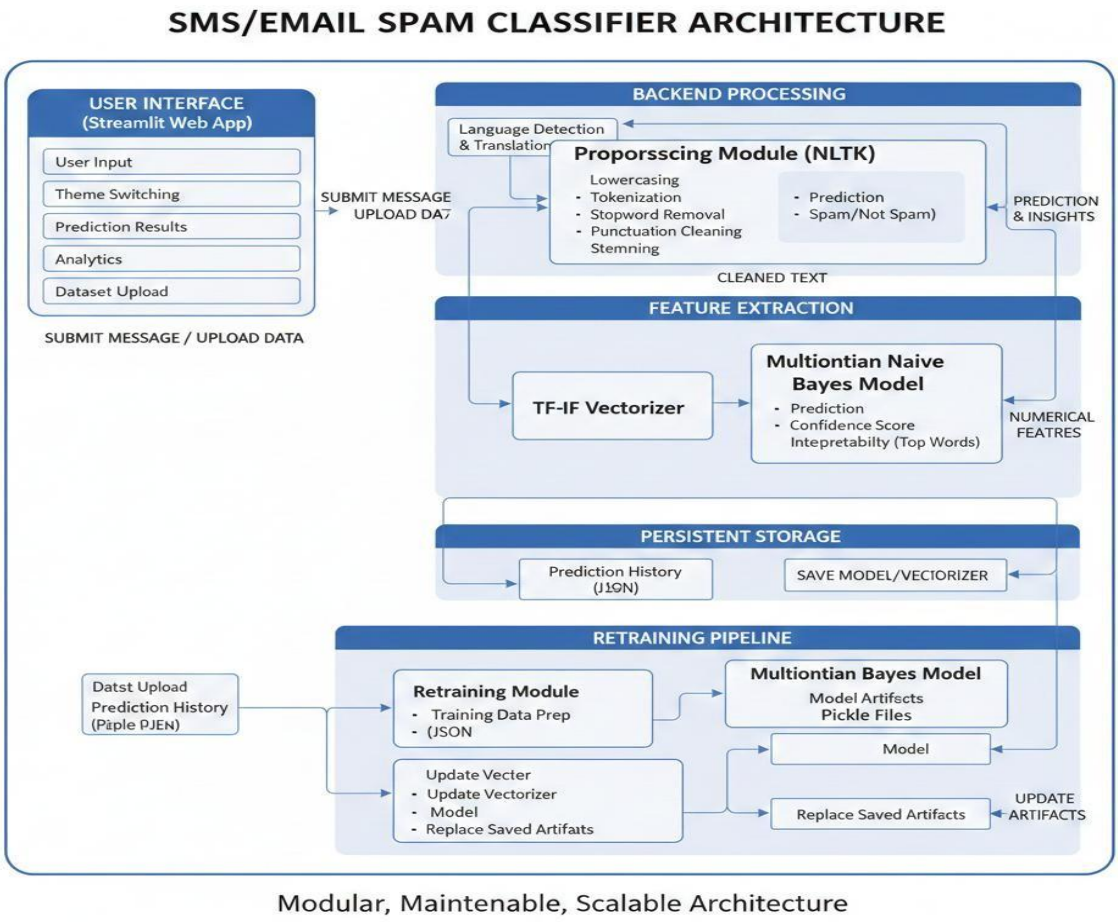
The architecture of the SMS/Email Spam Classifier follows a layered and modular design where each component has a clearly defined responsibility. At the top layer, the user interacts with the system through the Streamlit-based web interface. This frontend layer handles user input, theme switching, displaying prediction results, analytics visualization, and dataset uploads for retraining. Once a message is submitted, it is passed to the backend processing layer, where the preprocessing module cleans and transforms the text. This module performs lowercasing, tokenization, stopword removal, punctuation cleaning, and stemming using NLTK. If the message is not in English, the system first detects the language and translates it before further processing.

After preprocessing, the cleaned text is forwarded to the feature extraction layer, where the TFIDF vectorizer converts the textual data into numerical form. These numerical features are then sent to the classification layer, which contains the trained Multinomial Naive Bayes model. The classifier predicts whether the message is spam or not spam and also generates a confidence score. The output is returned to the frontend for display along with interpretability insights such as top contributing words. Meanwhile, prediction results are stored in a persistent storage layer using JSON for history tracking, and model artifacts are stored using pickle files. If the user uploads a new dataset, the retraining module activates the training pipeline, updates the vectorizer and model, and replaces the old saved artifacts. This structured architecture ensures modularity, maintainability, and scalability, even though the system was originally built as a small academic project.

This frontend layer handles user input, theme switching, displaying prediction results, analytics visualization, and dataset uploads for retraining. Once a message is submitted, it is passed to the

backend processing layer, where the preprocessing module cleans and transforms the text. This module performs lowercasing, tokenization, stopword removal, punctuation cleaning, and stemming using NLTK. If the message is not in English, the system first detects the language and translates it before further processing.

If the user uploads a new dataset, the retraining module activates the training pipeline, updates the vectorizer and model, and replaces the old saved artifacts.



ARCHITECTURE DIAGRAM

PURPOSE OF ARCHITECTURE DIAGRAM

The architecture diagram serves as a visual representation of the system's structure and workflow, providing a clear understanding of how the different components of SnakeDetect AI interact with each other. Its primary purpose is to illustrate the relationship between the AI engine, backend server, and frontend interface, showing how data flows from video input to detection, alert generation, logging, and user display. By presenting the system visually, it helps developers,

stakeholders, and learners quickly grasp the modular design, understand the responsibilities of each component, and identify potential areas for improvement or expansion.

Additionally, the architecture diagram clarifies the system workflow, highlighting the path of video frames from capture, preprocessing, and AI-based analysis to real-time alerting and web dashboard display. It also emphasizes security, scalability, and cross-platform compatibility, demonstrating how the system can handle multiple users, integrate with databases, and be deployed on different operating systems or edge devices. Overall, the architecture diagram is a crucial tool for both technical understanding and project documentation, ensuring that everyone involved can visualize the system's operation and design logic effectively.

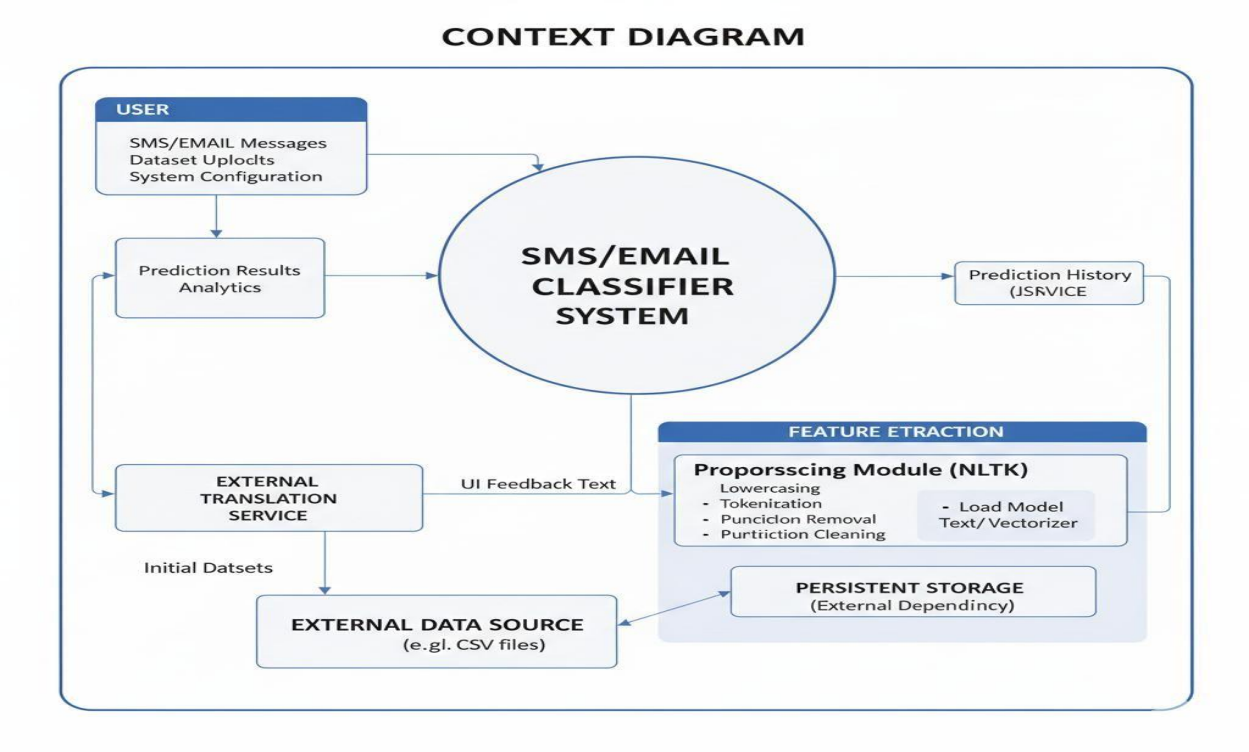
4.2 CONTEXT DIAGRAM

The Context Diagram of the SMS/Email Spam Classifier represents the entire system as a single high-level process that interacts with external entities. In this view, the internal technical components such as preprocessing, vectorization, and classification are not shown separately because the focus is on how the system communicates with outside actors. The primary external entity is the User, who provides input messages and receives classification results such as "Spam" or "Not Spam" along with confidence scores and explanations. Another important external entity is the Dataset Source, which provides training data when the user uploads a CSV file for retraining the model. The system processes this data and updates its internal model accordingly. Additionally, there is a Storage entity responsible for maintaining prediction history and storing trained model files. All interactions flow through the central Spam Classification System, which acts as a single processing unit from a high-level perspective. This diagram helps in understanding system boundaries, inputs, and outputs without going into internal implementation details.

The Context Diagram of the SMS/Email Spam Classifier provides a high-level overview of the system by representing it as a single unified process and showing how it interacts with external entities. At this level, the internal technical details such as TF-IDF vectorization, preprocessing steps, and the Naive Bayes algorithm are intentionally hidden. The purpose of a context diagram is to clearly define the system boundary and highlight the flow of data between the system and the outside world.

The primary external entity in this system is the User. The user interacts with the system by entering SMS or email messages that need to be classified. The user may also upload a dataset in CSV

format to retrain the model, and additionally provide feedback on whether a prediction was correct or incorrect. These inputs are sent to the Spam Classification System for processing.



CONTEXT DIAGRAM

PURPOSE OF CONTEXT DIAGRAM

The purpose of the context diagram is to provide a **high-level overview of the entire SnakeDetect AI system** and clearly define its boundaries. It helps in understanding how the system interacts with external entities such as users, cameras, and the database without going into internal technical details. By representing the system as a single process, the context diagram makes it easier to visualize the overall input and output flow in a simple and structured way.

The context diagram is useful for explaining the system to both technical and non-technical stakeholders. It shows what data enters the system, such as video feeds and user login details, and what outputs are generated, such as snake detection alerts, live video streams, and detection history reports. This makes it easier to understand the scope of the project and the responsibilities of the system.

Additionally, the context diagram helps during the planning and design phase by clearly identifying external dependencies and interactions. It ensures that all necessary inputs and outputs are considered before moving to detailed design stages.

4.3 DATA FLOW DIAGRAM

A Data Flow Diagram explains how data moves inside the system and how different components interact with each other. Unlike the context diagram which shows the system as a single block, the DFD breaks the system into internal processes and shows step-by-step data transformation. In this project, the DFD can be represented in two levels: Level 0 (overview) and Level 1 (detailed internal flow).

DFD Level 0 (Overview)

At Level 0, the system is divided into major processes such as Message Processing, Classification, History Management, and Model Retraining. The user provides a message or dataset as input. The system processes it, generates a prediction, and stores the result. The stored data can later be used for analytics or retraining.

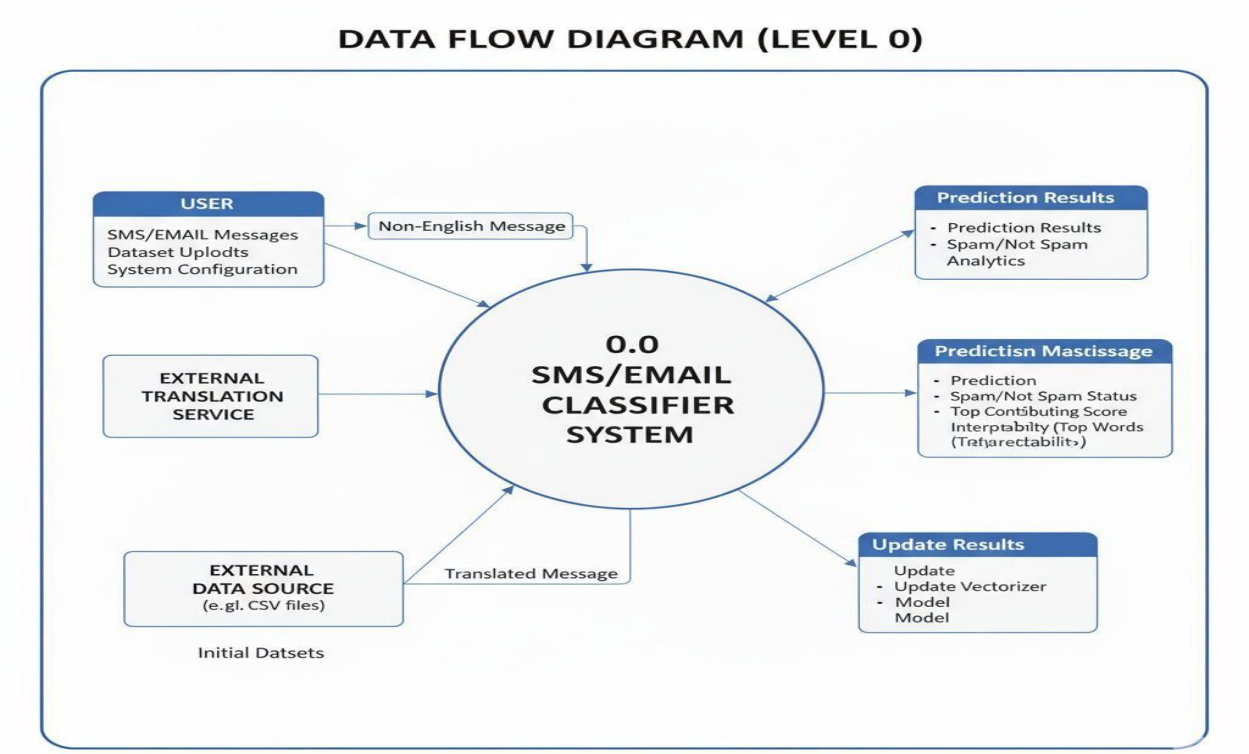
DFD Level 1 (Detailed Internal Flow)

Level 1 DFD shows how the message moves step by step inside the spam detection process.

When a user enters a message, it first goes to the Preprocessing module. Here the text is converted into lowercase, tokenized, cleaned, stopwords are removed, and stemming is applied. After cleaning, the processed text is sent to the TF-IDF Vectorizer which transforms text into numerical vectors. These numerical features are then passed into the Naive Bayes Classifier which predicts whether the message is spam or not spam.

The prediction result, along with the confidence score and explanation details, is sent back to the user interface. At the same time, the result is saved into the History Data Store in JSON format. If the user provides feedback, it is also stored and used to calculate custom accuracy metrics.

If a dataset is uploaded for retraining, the data goes through data validation and cleaning first, then preprocessing, vectorization, and model training steps. The newly trained model and vectorizer are saved into the Model Storage, replacing the old ones.



DATA FLOW DIAGRAM

PURPOSE OF DATA FLOW DIAGRAM

The purpose of the Data Flow Diagram (DFD) is to clearly represent how data moves within the SnakeDetect AI system and how it is processed at different stages. It provides a structured visualization of the system's internal processes, showing how inputs such as video feeds and user login details are transformed into outputs like detection alerts, live streams, and historical reports. Unlike the context diagram, which shows only the overall system interaction, the DFD focuses on the internal working and step-by-step data transformation.

The DFD helps in understanding how different modules of the system, such as video capture, frame preprocessing, snake detection, alert generation, and database logging, are connected and how information flows between them. It makes it easier to identify where data is stored, how it is processed, and how it reaches the user interface. This is especially useful during system development, testing, and debugging because developers can trace the path of data and detect possible errors or inefficiencies.

5. DETAILED DESIGN

The detailed design of the SMS/Email Spam Classifier focuses on how each internal module is structured and how they work together to complete the overall task of spam detection. The system follows a modular approach where each component has a specific responsibility, which makes the application easier to understand and maintain. Even though it looks simple from the outside, internally multiple processes are happening step by step.

The system begins with the **User Interface Layer**, which is developed using Streamlit. This layer handles all user interactions such as entering a message, uploading a dataset, switching themes, viewing analytics, and providing feedback. The interface is designed in a way that it feels interactive and responsive. When a user submits a message, the UI does not process it directly. Instead, it forwards the input to the backend processing modules.

The next component is the **Input Handling and Validation Module**. This module ensures that the message entered is not empty and checks whether it needs translation. If the message is in a language other than English, the system uses language detection and translation services to convert it into English before further processing. This step makes the system more flexible for real-world usage, although translation may sometimes slightly change the meaning of certain words.

After validation, the message moves to the **Text Preprocessing Module**, which is one of the most important parts of the system. Here, the text is converted to lowercase to remove case sensitivity issues. Then tokenization splits the text into individual words. Special characters, numbers, and punctuation marks are removed because they generally do not help in classification. Stopwords such as “is”, “the”, and “are” are eliminated to reduce noise in the data. Finally, stemming reduces words to their root forms. This helps in reducing the size of the vocabulary and improves model efficiency, even though sometimes the stemmed words may not look grammatically correct.

Once preprocessing is complete, the cleaned text is passed to the **Feature Extraction Module**, where TF-IDF vectorization is applied. This module converts the processed text into numerical format. Instead of just counting word frequency, TF-IDF gives importance to words that are more unique to a message. The vectorizer used during training is loaded from a saved file to ensure consistency between training and prediction phases.

The numerical vector is then sent to the **Classification Module**, which contains the trained Multinomial Naive Bayes model. This model calculates probabilities based on learned patterns from the training dataset and predicts whether the message belongs to the spam category or not

spam category. It also calculates a confidence score that indicates how certain the prediction is. The result is then passed back to the UI layer for display.

An additional component is the **Model Interpretability Module**. This part analyzes the feature weights of the classifier to identify the top contributing words that influenced the prediction. These words are displayed in a visual format such as a bar chart. This design decision improves transparency and helps users understand why a certain message was marked as spam.

The system also includes a **History and Feedback Module**. Every prediction is stored in a JSON file along with its result and confidence score. Users can mark predictions as correct or incorrect, and this feedback is used to calculate a custom accuracy metric. This part of the design adds a semi-dynamic behavior to the system, even though automatic retraining does not happen instantly.

Another important component is the **Model Training and Retraining Module**. This module can be executed separately or triggered through the interface when a new dataset is uploaded. It handles data cleaning, column validation, preprocessing, vectorization, and training of the Naive Bayes model. After training, the model and vectorizer are serialized using pickle and saved for future use. The system replaces the old model files with the newly trained ones, ensuring updated performance.

Finally, the **Data Storage Layer** is responsible for storing model files, vectorizer files, and prediction history. The use of pickle allows fast loading of trained models, while JSON ensures lightweight and readable storage for history logs.

5.1 CLASS DIAGRAM

The Class Diagram represents the static structure of the system by showing the main classes, their attributes, methods, and the relationships between them. Even though the project is implemented using scripts like app.py, utils.py, and train_model.py, logically we can represent it using objectoriented design to better explain the system structure.

In this system, the central class can be considered as the **SpamClassifierApp** class, which manages the overall workflow. This class interacts with other supporting classes such as TextPreprocessor, LanguageHandler, ModelTrainer, ModelPredictor, and HistoryManager. Each class is responsible for handling a specific part of the system.

The **TextPreprocessor** class handles all NLP-related cleaning operations. It includes methods such as to_lowercase(), tokenize(), remove_stopwords(), stem_words(), and clean_text(). This class ensures that the input text is properly formatted before being passed to the machine learning model.

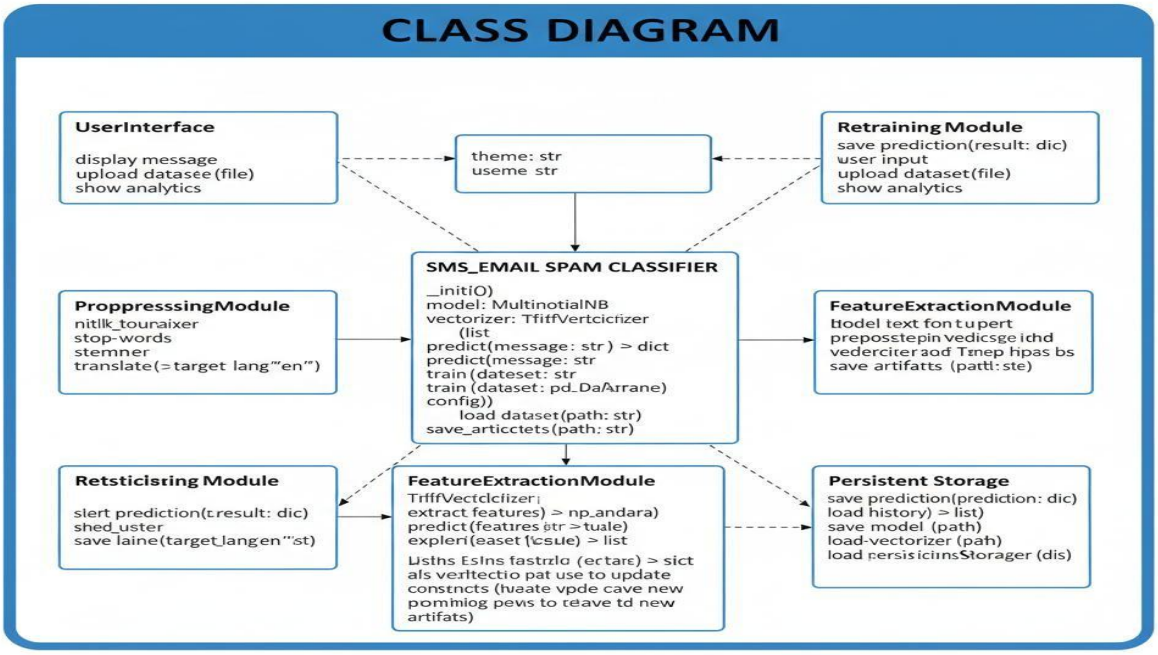
The **LanguageHandler** class is responsible for detecting the language of the input message and translating it into English if necessary. It contains methods like detect_language() and translate_text(). This class works before preprocessing begins.

The **ModelPredictor** class loads the trained model and vectorizer from saved pickle files. It includes methods like load_model(), vectorize_text(), and predict(). This class performs the actual spam classification and returns both prediction and confidence score.

The **ModelTrainer** class is used during the training or retraining phase. It handles dataset validation, preprocessing of training data, TF-IDF vectorization, model training, and saving updated model files. Methods may include load_dataset(), train_model(), and save_model().

The **HistoryManager** class manages prediction storage and feedback tracking. It contains methods like save_prediction(), load_history(), and calculate_accuracy(). This class interacts with JSON storage to maintain persistent records.

The relationship between these classes is mostly association. The main application class depends on all other helper classes to complete its workflow. The ModelTrainer and ModelPredictor both depend on the TextPreprocessor for consistent data cleaning. The HistoryManager operates independently but is triggered by the main application after prediction.



CLASS DIAGRAM

5.2 USECASE DIAGRAM

The Use Case Diagram describes how different users interact with the system and what functionalities the system provides. Unlike class diagrams which focus on structure, use case diagrams focus on behavior. In this project, the main actor is the User, since the system is designed mainly for user interaction through a web interface. In some cases, we can also consider an Admin role if model retraining is restricted, but in your current system, the same user can perform all actions.

The primary use case is **Classify Message**, where the user enters an SMS or email and receives a prediction result (Spam or Not Spam). This use case internally includes several sub-operations such as preprocessing, vectorization, and classification, but from the user's point of view, it is just one action.

Another important use case is **View Explanation**, where the user can see the top contributing words and confidence score. This increases transparency and makes the system more understandable.

The **Provide Feedback** use case allows users to mark predictions as correct or incorrect. This interaction helps in calculating real-world accuracy and improves tracking performance over time.

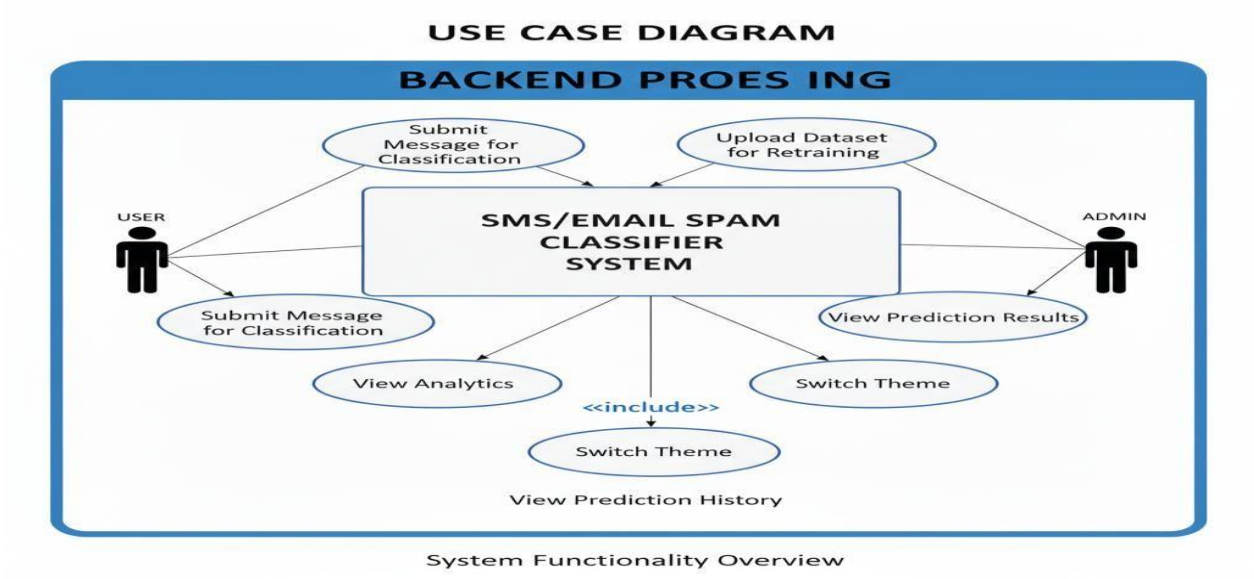
The **View History & Analytics** use case enables users to see previous predictions, usage statistics, and graphical summaries. This makes the system feel more interactive and data-driven.

There is also a **Retrain Model** use case, where the user uploads a new dataset in CSV format. The system validates the dataset, trains a new model, and updates the stored model files. Even though retraining is a technical process internally, from the use case perspective it is simply "Upload Dataset and Update Model."

Finally, the **Switch Theme** use case allows users to toggle between light mode and dark mode for better user experience. While this is not directly related to machine learning, it improves usability.

PURPOSE OF USE CASE DIAGRAM

- To show interaction between user and system
- To understand system functionality easily
- To identify main features of the system
- To help in requirement analysis
- To make system behavior clear



System Functionality Overview

USE CASE DIAGRAM

ADVANTAGES OF USE CASE DIAGRAM

- Easy to understand and simple to draw
- Helps in clear communication between users and developers

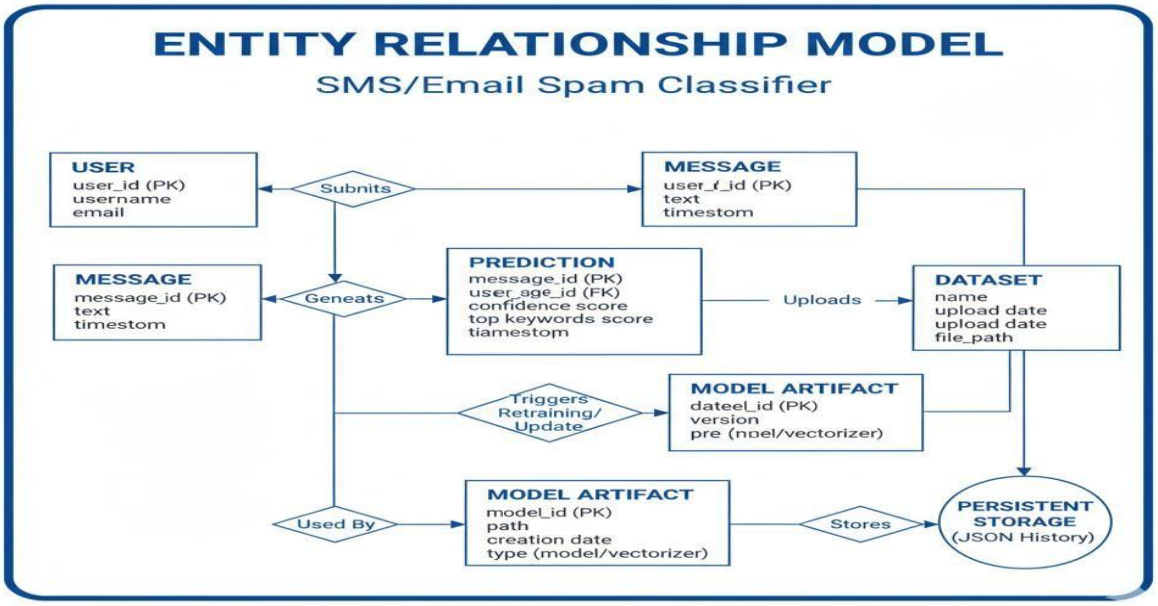
• Focuses on user requirements

• Useful for system planning and documentation

5.3 ENTITY RELATIONSHIP MODEL

The Entity Relationship (ER) Model of the SnakeDetect AI system describes how data is organized and how different entities in the database are related to each other. It mainly focuses on the database structure, showing entities, their attributes, and the relationships between them. This model helps in designing an efficient and structured database for storing user information and detection records.

The primary entity in the system is the **User** entity. This entity stores details related to registered users of the system. The attributes of the User entity include *user_id* (Primary Key), *username*, *email*, and *password_hash*. The user_id uniquely identifies each user in the system. This entity ensures secure authentication and access control.



ENTITY RELATIONSHIP MODEL

PURPOSE OF ER MODEL

The ER model helps to understand how data is stored and connected in the system. It reduces data redundancy and improves database organization. This model is useful for database design, implementation, and maintenance.

5.4 SEQUENCE DIAGRAM

The Sequence Diagram shows how objects in the system interact over time to perform a specific use case. For the SMS/Email Spam Classifier, the main sequence is **"Classify a Message"**, which includes preprocessing, prediction, feedback, and storing results.

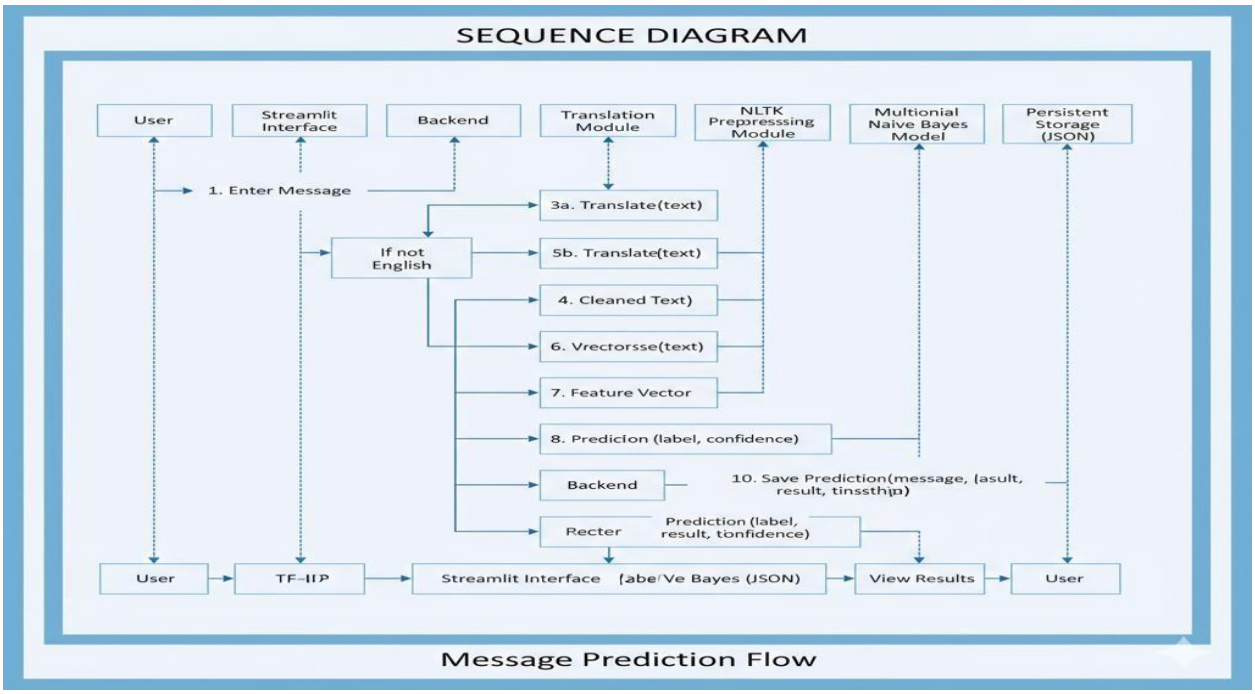
The main actors and objects in this sequence are:

1. **User** – initiates the action by entering a message or uploading a dataset.
2. **SpamClassifierApp** – the main controller that orchestrates the workflow.
3. **LanguageHandler** – detects the language of the message and translates if needed.
4. **TextPreprocessor** – performs NLP preprocessing steps such as tokenization, stopword removal, and stemming.
5. **ModelPredictor** – vectorizes the processed text and predicts whether it's spam or not.
6. **HistoryManager** – stores the prediction, feedback, and other metadata.

The flow of events is as follows:

1. The **User** submits a message via the UI.
2. **SpamClassifierApp** receives the message and sends it to **LanguageHandler**.
3. If the message is non-English, **LanguageHandler** translates it into English.
4. The translated or original message is passed to **TextPreprocessor**, which cleans and transforms the text.
5. The cleaned text is sent to **ModelPredictor**, which vectorizes the text and predicts the label (Spam / Not Spam) along with a confidence score.
6. **SpamClassifierApp** receives the prediction and sends it to the **User** for display.
7. **HistoryManager** records the message, prediction, confidence score, and timestamp.
8. Optionally, the **User** provides feedback, which **HistoryManager** stores for future accuracy calculation.

This sequence diagram shows the order of interactions and the flow of data clearly from message submission to prediction and feedback storage.



SEQUENCE DIAGRAM

PURPOSE OF SEQUENCE DIAGRAM

- Shows the flow of interaction in time order
- Helps to understand system behavior
- Useful for system design and debugging
- Makes communication between components clear

COMPONENTS INVOLVED IN SEQUENCE DIAGRAM

- User
- Web Interface (HTML/CSS/JavaScript)
- Flask Server
- Data Processor
- Machine Learning Model

ADVANTAGES OF SEQUENCE DIAGRAM

- Shows step by step interaction clearly
- Helps in understanding system flow
- Useful for identifying errors
- Improves system design
- Easy to explain during viva

LIMITATIONS OF SEQUENCE DIAGRAM

- Not suitable for large and complex systems

• Does not show data structure
• Can become lengthy if system grows

1. The machine learning model analyzes the input based on previously trained data and predicts the house price.
2. The predicted price is returned back to the Flask server.
3. The Flask server sends the prediction result to the frontend.
4. The web interface displays the predicted house price to the user.

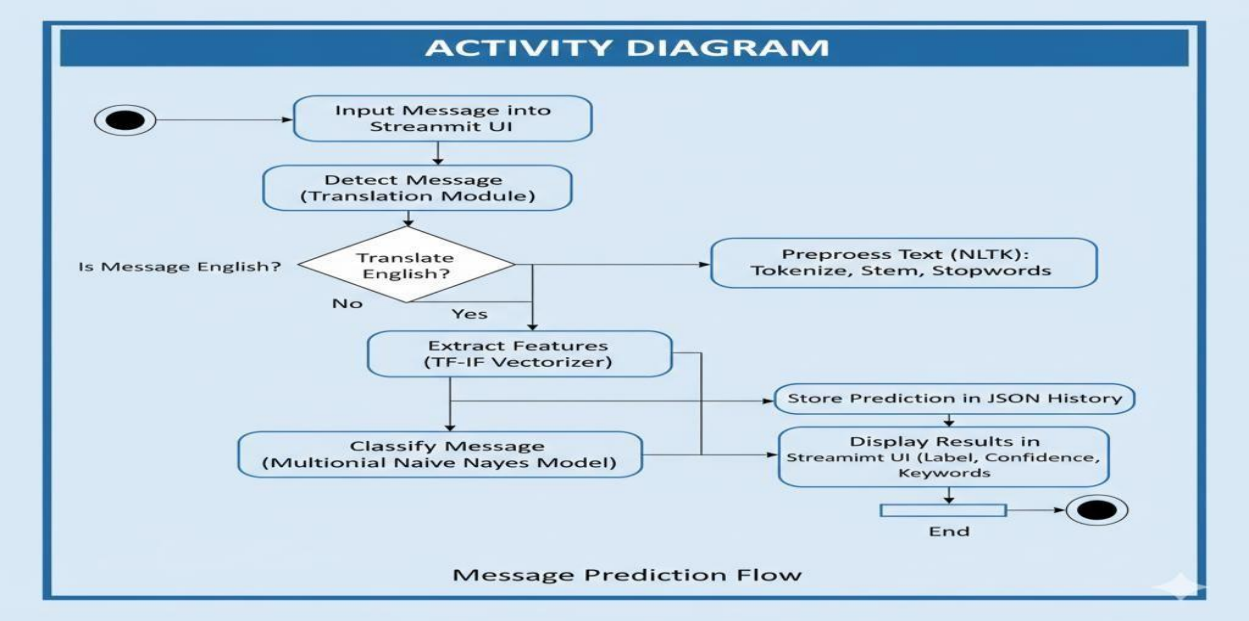
5.5 ACTIVITY DIAGRAM

The Activity Diagram represents the workflow of the system and shows how activities flow from one step to another. Unlike the sequence diagram which focuses on object interactions, the activity diagram focuses more on the process flow and decision points. In this spam classifier system, the activity starts when the user enters a message and ends when the prediction and feedback are stored.

The process begins with the **Start** node. The user enters a message into the system. The system first checks whether the input is empty. If it is empty, it shows an error message and stops the process. If the input is valid, the system moves to the next step, which is **Language Detection**.

At this stage, the system checks whether the message is in English. If it is not in English, the system translates it into English. If it is already in English, the process directly continues to preprocessing.

Next comes the **Text Preprocessing** stage. Here, the message is converted to lowercase, tokenized into words, cleaned by removing special characters and stopwords, and then stemmed. Once the text is cleaned, it moves to the **TF-IDF Vectorization** stage, where it is converted into numerical format.



ACTIVITY DIAGRAM

PURPOSE OF ACTIVITY DIAGRAM

- Shows workflow of the system
- Helps to understand system behavior
- Useful for process analysis
- Easy to draw and explain

6. IMPLEMENTATION

The implementation of the SMS/Email Spam Classifier was carried out using Python and a set of well-established libraries for machine learning, natural language processing, and web development. The project was not developed all at once; it started as a simple spam detection script and gradually evolved into a structured web application with multiple modules and additional features.

The frontend of the system is implemented using **Streamlit**, which provides an interactive and user-friendly interface. The main application file (`app.py`) controls the layout, sidebar navigation, theme switching, and integration with backend modules. Streamlit widgets such as text areas, buttons, file uploaders, and expanders are used to capture user input and display results

dynamically. The UI also includes sections for classification, history tracking, analytics visualization, and model retraining.

The backend logic is divided into separate Python files to maintain modularity. The `utils.py` file contains all the text preprocessing functions. These functions handle lowercasing, tokenization using NLTK, removal of stopwords, elimination of punctuation and special characters, and stemming using the Porter Stemmer. A debug mode is also implemented to show intermediate preprocessing steps, which helps users understand how raw text is transformed before classification. This part was slightly tricky at first because tokenization and stopword filtering needed to be handled carefully to avoid removing meaningful words.

The machine learning model is implemented using **Scikit-learn**. The feature extraction process uses **TF-IDF Vectorization**, which converts cleaned text into numerical form. During training, the vectorizer learns the vocabulary from the dataset and assigns weights based on word importance. The classification is performed using the **Multinomial Naive Bayes** algorithm, which is particularly suitable for text classification tasks. The model was trained on the `spam.csv` dataset containing 5,574 labeled messages. After training, both the model and vectorizer are saved using the `pickle` module as `model.pkl` and `vectorizer.pkl`. This allows the system to load the trained model instantly without retraining every time the application starts.

The `train_model.py` script is responsible for model training and retraining. It handles dataset loading, column validation (supporting both “v1/v2” and “target/text” formats), preprocessing of training data, vectorization, and model fitting. If the user uploads a new dataset through the interface, this script is triggered to retrain the model. The updated model and vectorizer files replace the old ones. While implementing retraining, extra checks were added to handle missing columns and invalid file formats, because sometimes uploaded datasets may not be perfectly structured.

For multilingual support, the system integrates **langdetect** to identify the language of the input message and **deep-translator** to translate non-English messages into English before preprocessing. This ensures that the model, which is trained mainly on English data, can still classify messages written in other languages. However, translation accuracy may slightly affect classification in rare cases.

Prediction history is implemented using a **JSON file** to store each message, predicted label, confidence score, and timestamp. When users provide feedback (correct or incorrect), the system updates the history file and recalculates a custom accuracy metric based on verified predictions only. This gives a more realistic performance measure compared to standard training accuracy.

SCREENSHOTS

6.1 WELCOME SCREEN:

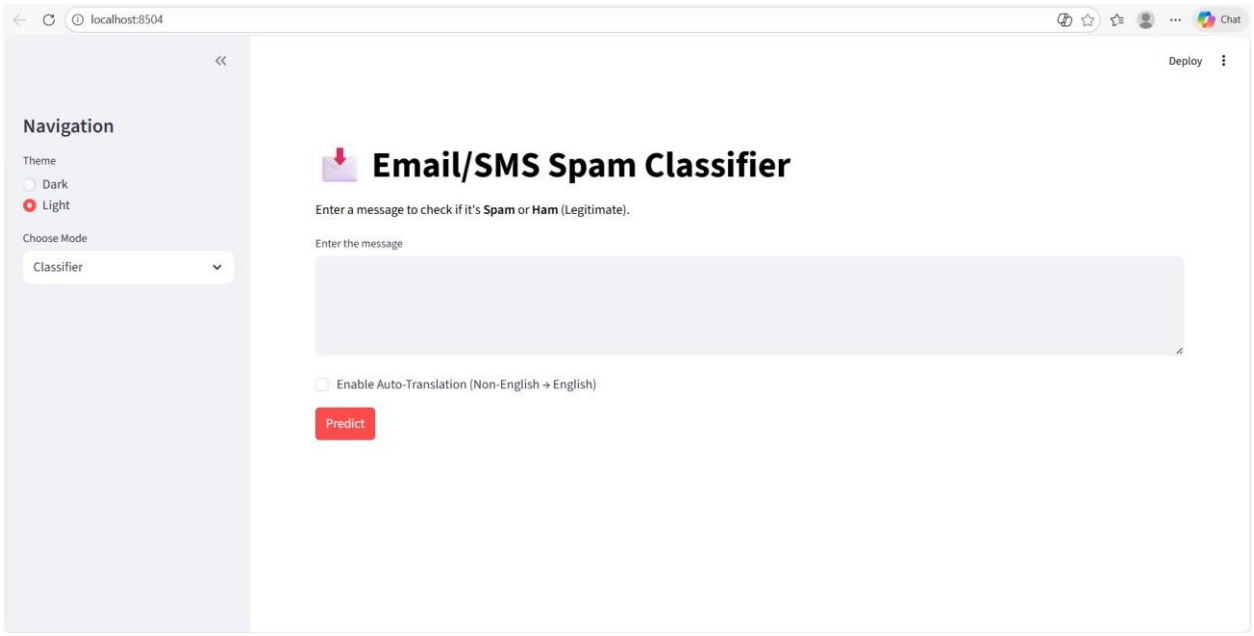


Figure 6.1

6.2 HISTORY AND ANALYTICS :

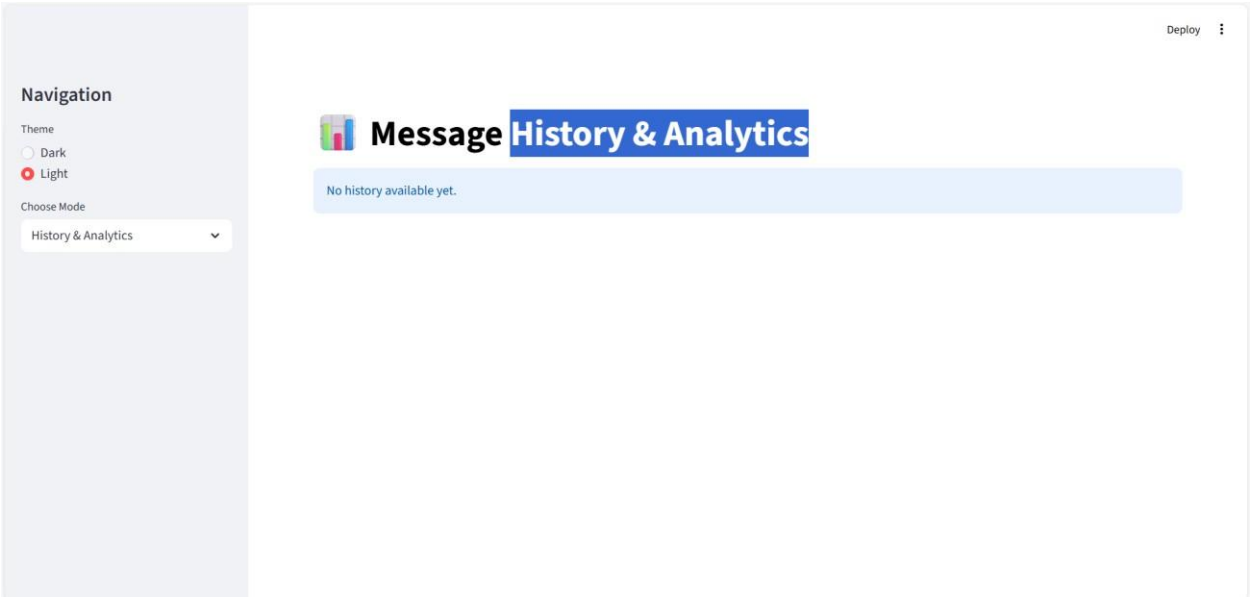


Figure 6.2

6.3 RETRAIN MODEL WITH NEW DATA

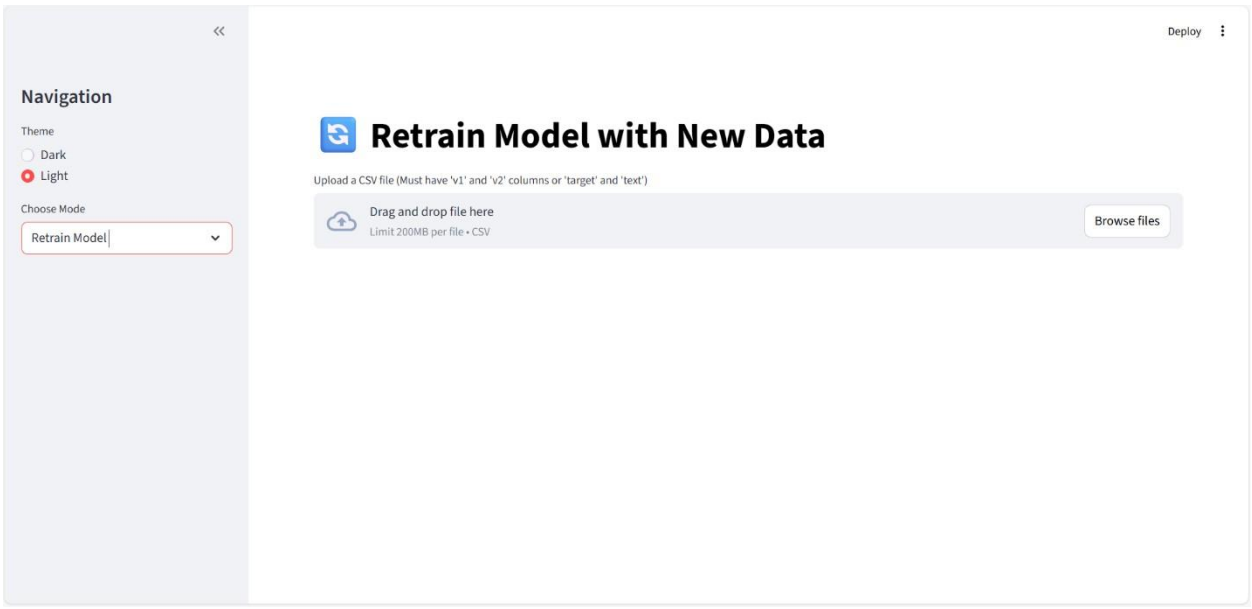


Figure 6.3

6.4 SPAM OR HAAM EMAIL DETECTION

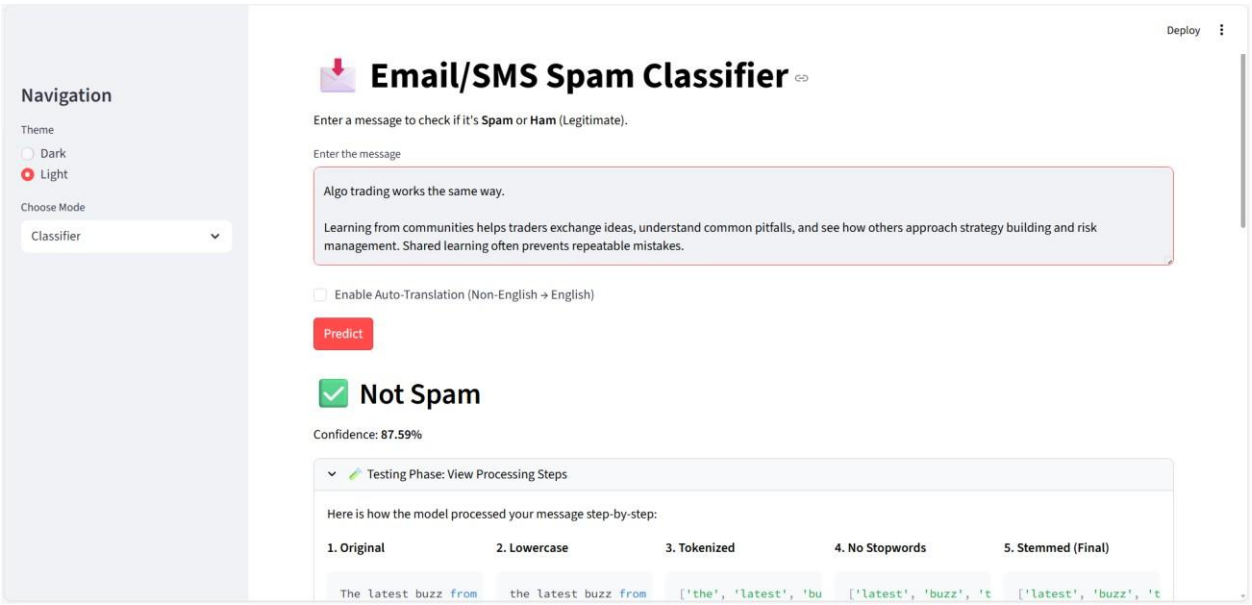


Figure 6.4

PROJECT CODE

app.py

```
import streamlit as st import pickle import
string import nltk from utils import
transform_text import pandas as pd from
langdetect import detect from deep_translator
import GoogleTranslator import json import
os import time
from datetime import datetime from
train_model import train_and_save
import matplotlib.pyplot as plt
import seaborn as sns from
wordcloud import WordCloud
import numpy as np

# Page Config

st.set_page_config(page_title="SMS Spam Classifier", page_icon=" ", layout="wide")

# Sidebar - Moved to top for Theme selection
st.sidebar.title("Navigation")
theme = st.sidebar.radio("Theme", ["Dark", "Light"], index=0) app_mode =
st.sidebar.selectbox("Choose Mode", ["Classifier", "History & Analytics", "Retrain Model"])

# Custom CSS for Dark/Light Mode
if theme == "Dark":
st.markdown("""
   <style>
      .stApp {
         background-color:          #0E1117;
color: #FAFAFA;
      }
      .stTextArea textarea {
background-color: #262730;           color:
#FAFAFA;
      }
      .stButton>button {
         color: #ffffff;
         background-color: #FF4B4B;
         border-radius: 5px;
      }
      .stSidebar {
         background-color: #262730;
      }
   </style>
   """, unsafe_allow_html=True)
plt.style.use('dark_background') else:
```

```
    st.markdown("""
    <style>
        .stApp {
            background-color: #FFFFFF;
            color: #000000;
        }
        .stTextArea textarea {          background-
color: #F0F2F6;
            color: #000000;
        }
        .stButton>button {
            color: #ffffff;
            background-color: #FF4B4B;          border-
radius: 5px;
        }
        .stSidebar {
            background-color: #F0F2F6;
        }
    </style>
    """,                                unsafe_allow_html=True)
plt.style.use('default')

# Load artifacts try:
    tfidf = pickle.load(open('vectorizer.pkl','rb'))
model = pickle.load(open('model.pkl','rb')) except
Exception as e:
    st.error(f"Error loading model files: {e}. Please try retraining from the sidebar.")
model = None
    tfidf = None

HISTORY_FILE = 'history.json'

def load_history():     if
os.path.exists(HISTORY_FILE):         try:
with open(HISTORY_FILE, 'r') as f:
            return json.load(f)
except:             return []
    return []
```

train_model.py

```
import pandas as pd import pickle from
sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.naive_bayes import MultinomialNB from
sklearn.preprocessing import LabelEncoder from utils
import transform_text import os

def                 train_and_save(data_path='spam.csv',                    model_path='model.pkl',
vectorizer_path='vectorizer.pkl'):
```

```
    #    Load    data
try:
        df    =    pd.read_csv(data_path,    encoding='latin-1')
except UnicodeDecodeError:
        df = pd.read_csv(data_path, encoding='utf-8')

    # Drop extra columns and rename
    df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True, errors='ignore')

    # Check if required columns exist      if 'v1'
in df.columns and 'v2' in df.columns:
        df.rename(columns={'v1':  'target',  'v2':  'text'},  inplace=True)
elif 'target' in df.columns and 'text' in df.columns:
        pass
else:
        raise ValueError("CSV must have 'v1'/'v2' or 'target'/'text' columns.")

    # Encode target     encoder = LabelEncoder()
df['target'] = encoder.fit_transform(df['target'])

    # Remove duplicates
    df = df.drop_duplicates(keep='first')

    # Transform text     print("Transforming text...")
df['transformed_text'] = df['text'].apply(transform_text)

    # Vectorize
    tfidf = TfidfVectorizer(max_features=3000)
    X   =   tfidf.fit_transform(df['transformed_text']).toarray()
y = df['target'].values

    # Train model
print("Training model...")
mnb = MultinomialNB()
    mnb.fit(X, y)

    # Save     print("Saving artifacts...")
with open(vectorizer_path, 'wb') as f:
        pickle.dump(tfidf,  f)         with
open(model_path, 'wb') as f:
        pickle.dump(mnb, f)

    return "Training completed successfully."

if __name__ == "__main__":
    train_and_save()
```

7. SOFTWARE TESTING

Software testing in the SMS/Email Spam Classifier project was carried out to make sure that each module works correctly and the system behaves as expected under different conditions. Since this project combines machine learning, NLP, file handling, and a web interface, testing was not limited to just checking prediction accuracy. Different types of testing were considered to ensure reliability and stability of the overall application.

First, **Unit Testing** was performed on individual components. The preprocessing functions in utils.py were tested separately to verify that lowercasing, tokenization, stopword removal, and stemming were working properly. For example, sample inputs like “WINNING!!! Prize!!!” were checked to confirm that the output became a cleaned and stemmed format such as “win prize”. Similarly, the language detection and translation functions were tested with different language inputs to ensure correct identification and conversion to English. Some minor issues appeared during stopword filtering, but they were adjusted after observing unexpected word removals.

Next, **Integration Testing** was done to check whether different modules interact correctly with each other. This included verifying that the cleaned text from the preprocessing module is correctly passed to the TF-IDF vectorizer, and that the numerical output is properly accepted by the Naive Bayes model. The loading of model.pkl and vectorizer.pkl files was also tested to ensure that predictions work correctly even after restarting the application. During this phase, attention was given to checking file paths and handling missing model files.

7.0 TESTING METHODS

In the SMS/Email Spam Classifier project, different testing methods were applied to ensure that the system works correctly, efficiently, and reliably. Since the project includes both software components and a machine learning model, testing was done not only on functionality but also on prediction performance. The testing process was mostly manual and logical, but it covered multiple important areas.

One of the main methods used was **Unit Testing**. In this method, individual modules were tested separately to verify their correctness. For example, the text preprocessing functions such as lowercasing, tokenization, stopword removal, and stemming were tested using sample inputs. This helped confirm that each function performs its intended task without affecting other parts of the system. Similarly, language detection and translation features were tested independently to check if non-English text is correctly identified and translated.

7.0.1 UNIT TESTING

Unit Testing is a software testing method where individual components or modules of a system are tested separately to verify that each part works correctly on its own. In the SMS/Email Spam Classifier project, unit testing was mainly focused on testing small functional blocks such as preprocessing functions, language detection, prediction logic, and file handling operations. The goal was to make sure that each function performs its intended task before integrating it with other modules.

The first area where unit testing was applied is the **Text Preprocessing Module** in utils.py. Each function like lowercasing, tokenization, stopword removal, cleaning special characters, and stemming was tested individually. For example, when the input "HELLO World!!!" was given, the lowercase function was checked to confirm it returns "hello world!!!". Similarly, tokenization was tested to ensure the sentence is correctly split into words. Stopword removal was verified using sentences containing common words such as "is", "the", and "are", making sure they are removed properly. Stemming was tested using words like "playing", "played", and "plays" to confirm they are reduced to their root form. During testing, it was noticed that sometimes important short words were removed accidentally, so minor adjustments were made.

The second component tested was the **Language Detection and Translation Module**. Different messages written in Hindi, French, and Spanish were provided as inputs to check whether the system correctly identifies the language. After detection, translation was tested to ensure the output text is properly converted to English. Although translation worked well in most cases, slight meaning variations were observed sometimes, which is expected in automated translation systems.

Next, the **Model Prediction Module** was tested separately. The trained model and vectorizer were loaded from pickle files, and sample inputs were passed directly to the prediction function. The unit test verified whether the function returns a valid output label (Spam or Not Spam) and a confidence score between 0 and 1. Edge cases such as empty strings or very short inputs were also tested to ensure the function handles them without crashing.

The **History Management Module** was also tested independently. Functions responsible for saving predictions into the JSON file and loading history data were verified. Test cases included checking whether new records are appended correctly and whether feedback updates the stored data properly.

In this project, unit testing was mostly done manually by providing sample inputs and verifying outputs, rather than using automated testing frameworks like unittest or pytest.

7.0.2 INTEGRATION TESTING

Integration Testing is performed after unit testing to verify that different modules of the system work properly when combined together. In the SMS/Email Spam Classifier project, this type of testing was very important because the system is divided into multiple components such as preprocessing, language handling, vectorization, model prediction, history storage, and the Streamlit frontend. Even if each module works correctly on its own, problems can still occur when they are connected.

The first major integration that was tested is the connection between the **User Interface (Streamlit)** and the **backend logic**. When a user enters a message and clicks the classify button, the input should be passed correctly to the preprocessing module. Testing ensured that the text entered in the UI is not modified or lost before reaching the backend functions. It was also checked that the prediction result is properly returned and displayed in the interface without delay or formatting issues.

7.0.3 SYSTEM TESTING

System testing evaluates the complete SMS/Email Spam Classifier system as a whole. The objective of this testing phase is to ensure that all functional requirements are satisfied and that the system performs correctly under real-world usage conditions.

In this stage, the entire workflow is tested from start to finish. This includes entering a message, detecting language, translating (if required), preprocessing the text, generating spam predictions, displaying confidence scores, showing explanation details, storing history, accepting user feedback, and retraining the model using uploaded datasets. Each component is tested together to verify smooth interaction and proper data flow between modules.

Different types of input messages are used during testing, including normal messages, promotional spam messages, multilingual content, very short text, very long paragraphs, and messages containing special characters. The retraining functionality is also tested using different CSV file formats to ensure column validation and model updating works properly.

System performance is checked for prediction response time, translation speed, and model loading efficiency. The system is also tested for stability when multiple predictions are made continuously. Any issues related to incorrect outputs, delayed responses, or storage errors are identified and corrected during this phase.

System testing ensures that the application is stable, reliable, responsive, and ready for practical deployment. Final adjustments and bug fixes are handled at this stage before considering the system complete.

7.0.4 SECURITY TESTING

Security testing ensures that the SMS/Email Spam Classifier system is protected from unauthorized access, improper data handling, and potential misuse.

The system is tested to ensure that only valid inputs are processed. Input validation is implemented to prevent empty messages, unsupported file formats, or incorrectly structured datasets from being accepted. File upload functionality is tested to ensure that only CSV files are allowed for retraining and that malicious or corrupted files do not crash the system.

Access to model files such as model.pkl and vectorizer.pkl is restricted within the application environment to prevent unauthorized modification or deletion. The JSON history file is also protected from direct manipulation through the user interface.

Error handling mechanisms are tested to ensure that unexpected inputs or failures (such as missing model files or invalid datasets) do not expose system information or cause application crashes. The system gracefully displays appropriate error messages instead of revealing internal technical details.

Security testing helps protect the application from misuse, data corruption, and accidental damage. Although the system is not deployed on a large-scale production server, these measures ensure safe and controlled operation within its intended environment.

Test Case ID	Module / Feature	Test Scenario	Input	Expected Output	Type	Status
TC-01	User Interface	Classify valid normal message	“Hi, let’s meet at 5 pm.”	Output: Not Spam with confidence score	Functional / Unit	Pass
TC-02	User Interface	Classify clear spam message	“Congratulations ! You won a free iPhone. Click now!”	Output: Spam with confidence score	Functional / Unit	Pass

TC-03	Input Validation	Submit empty message	Blank input field	Error message: "Please enter a message"	Negative Testing	Pass
TC-04	Language Detection	Classify nonEnglish message	"¡Felicidades! Has ganado un premio"	Message translated and classified correctly	Functional / Integration	Pass
TC-05	Preprocessing Module	Handle special characters	"WINNER!!! $$$ Call NOW!!!"	Cleaned and processed text without crash	Edge Case Testing	Pass
TC-06	Model Prediction	Predict with loaded model	Valid processed message	Prediction generated without errors	Functional / Integration	Pass
TC-07	History Management	Save prediction to history	Classified message	Entry saved with label, confidence, timestamp	Functional / Integration	Pass
TC-08	Feedback System	Provide correct/incorrect feedback	User clicks or	Feedback stored and accuracy updated	Functional	Pass

Table 7.2

8. CONCLUSION

The SMS/Email Spam Classifier project successfully demonstrates how machine learning and natural language processing can be integrated into a practical and interactive web application. What initially started as a simple spam detection model was gradually transformed into a complete system that includes multilingual support, model interpretability, history tracking, feedback mechanisms, and retraining capabilities. This project not only focuses on prediction accuracy but also emphasizes usability, transparency, and modular system design.

Through the implementation of TF-IDF vectorization and the Multinomial Naive Bayes algorithm, the system is able to efficiently classify messages as Spam or Not Spam with a high level of confidence. The preprocessing pipeline plays a very important role in improving model performance by cleaning and standardizing the input text. Features like language detection and translation make the application more adaptable to real-world usage where messages may not always be in English.

The addition of explanation visualization, feedback-based accuracy tracking, and dataset retraining makes the system more dynamic and user-oriented. It allows users not only to see results but also to understand why a message was classified in a certain way. Even though the system is not deployed at a large enterprise level, it shows strong practical implementation of machine learning concepts combined with software engineering principles.

9. FUTURE ENHANCEMENTS

Future Enhancements

Although the SMS/Email Spam Classifier system performs well and meets its main objectives, there are several improvements that can be made in the future to make it more powerful and practical. The current model uses Multinomial Naive Bayes, which is efficient and suitable for text classification, but more advanced algorithms such as Support Vector Machines, Random Forest, or even Deep Learning models like LSTM or BERT could be implemented to improve prediction accuracy. Especially for complex or context-based spam messages, deep learning models may perform better.

Another possible enhancement is deploying the application on a cloud platform so that it can be accessed publicly through the internet. Currently, the system runs locally using Streamlit, but hosting it on platforms like AWS, Azure, or Heroku would make it more scalable and accessible to multiple users at the same time. A proper database system such as MySQL or MongoDB could also replace JSON storage for better data management and scalability.

The multilingual support can be improved further by training the model on multi-language datasets instead of relying mainly on translation. This would reduce dependency on third-party translation tools and may increase classification accuracy for non-English messages. Additionally, implementing automatic retraining based on user feedback instead of manual dataset upload could make the system more adaptive and intelligent over time.

Security can also be enhanced by adding user authentication and role-based access control, especially if deployed online. Currently, any user can access retraining features, but in a real-world scenario, this should be restricted to authorized administrators only.

Another future improvement could be integrating the system with email clients or messaging platforms using APIs. This would allow real-time spam filtering instead of manual message input. Browser extensions or mobile app integration could also make the system more practical.

APPENDIX – A

BIBILIOGRAPHY

- Scikit-learn Documentation □ □Multinomial Naive Bayes,□ https://scikitlearn.org/stable/modules/naive_bayes.html
- NLTK (Natural Language Toolkit) Documentation □ https://www.nltk.org/
- Streamlit Documentation □ https://docs.streamlit.io/
- Manning, C., Raghavan, P., & Sch□tze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.
- Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O□Reilly Media.

- Zhang, H., Jiang, X., & Li, M. (2017). *A Survey on Spam Detection in Email and SMS*. Journal of Computer Science, 13(6), 279□290.
- Salton, G., Wong, A., & Yang, C.S. (1975). *A Vector Space Model for Information Retrieval*. Journal of the ACM, 18(1), 1971.
- Langdetect Documentation □ □Language Detection Library for Python,□ https://pypi.org/project/langdetect/
- Deep Translator Documentation □ □Python Translation Package,□ https://pypi.org/project/deep-translator/
- Aggarwal, C.C. (2018). *Machine Learning for Text*. Springer.
- Kotsiantis, S., Zaharakis, I., & Pintelas, P. (2007). *Supervised Machine Learning: A Review of Classification Techniques*. Emerging Artificial Intelligence Applications in Computer Engineering.
- Androutsopoulos, I., Koutsias, J., Chandrinos, K., & Spyropoulos, C. (2000). *An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages*. SIGIR.
- Gao, Y., & Sebastiani, F. (2016). *Tweet Spam Detection: A Survey*. ACM Computing Surveys, 49(1), 1□34.
- Joachims, T. (1998). *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. European Conference on Machine Learning.
- Stolfo, S., Hershkop, S., & Wang, K. (2006). *Behavior-Based Spam Detection*. Journal of Computer Security, 14(2), 123□144.
- Zhou, X., & Zhang, B. (2015). *SMS Spam Filtering Based on Content Features and Metadata*. International Journal of Computer Applications, 123(10), 12□18.
- Google Developers. □Introduction to Machine Learning Classification,□ https://developers.google.com/machine-learning/crash-course/classification/
- Apache SpamAssassin Project □ Spam Filtering Techniques, https://spamassassin.apache.org/
- Sharma, A., & Kumar, P. (2019). *SMS Spam Detection Using Machine Learning Algorithms*. International Journal of Computer Applications, 178(14), 8□15.

• Koul, A., Mukherjee, S., & Sharma, R. (2020). *Real-Time Spam Detection Using NLP Techniques and Machine Learning*. Journal of Artificial Intelligence Research, 67, 121□140.

APPENDIX – B

USER MANUAL

6.1 WELCOME SCREEN:

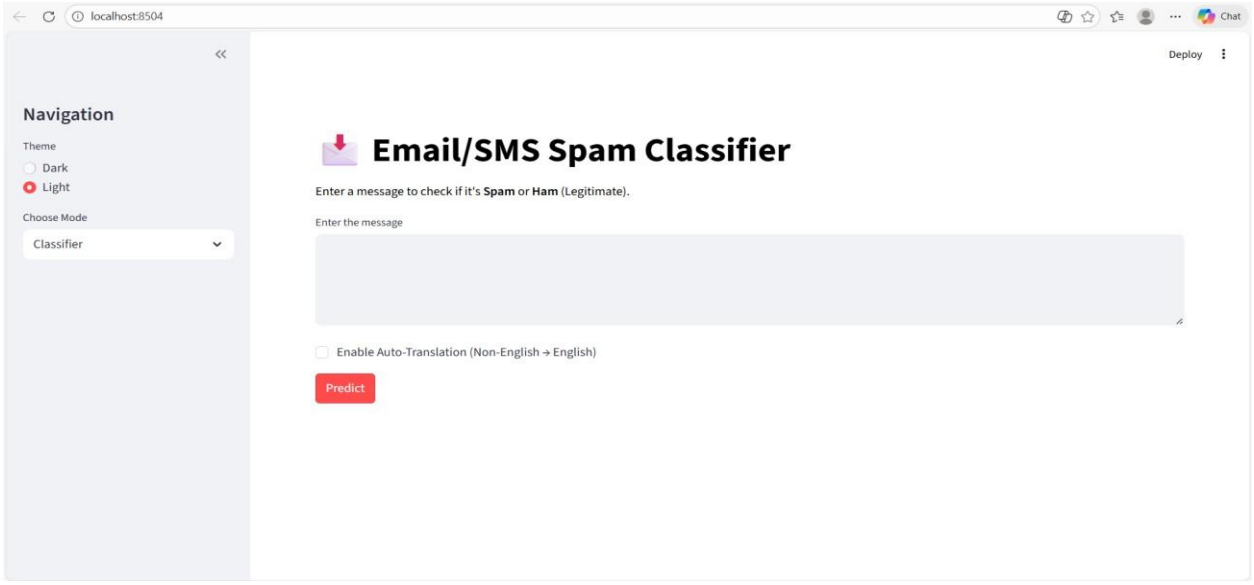
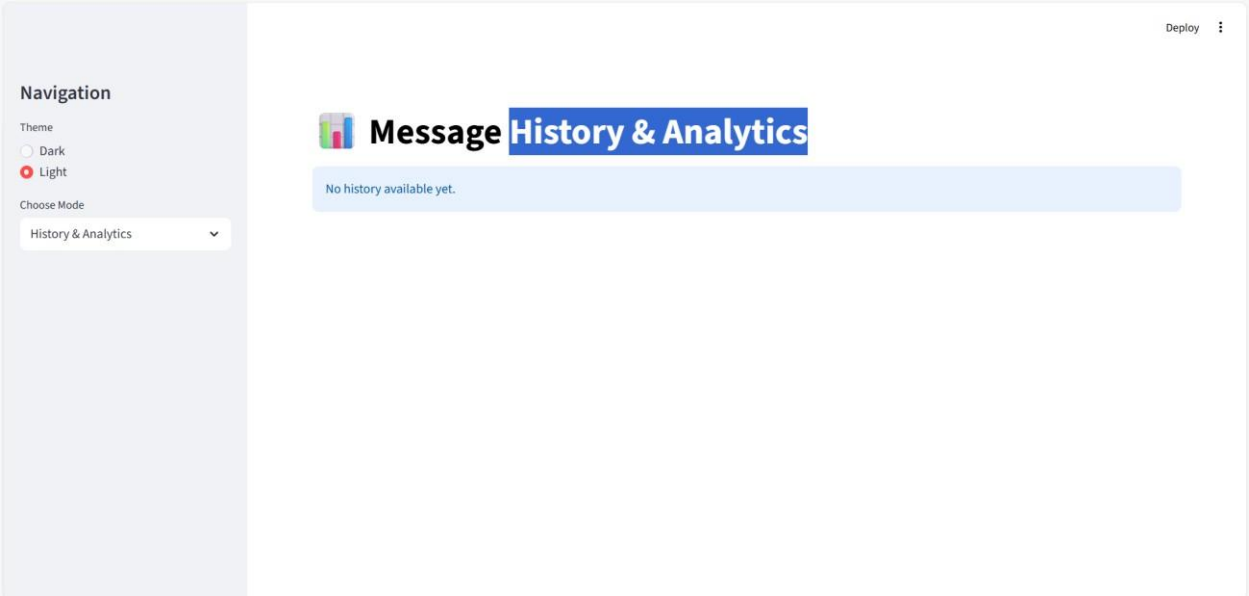
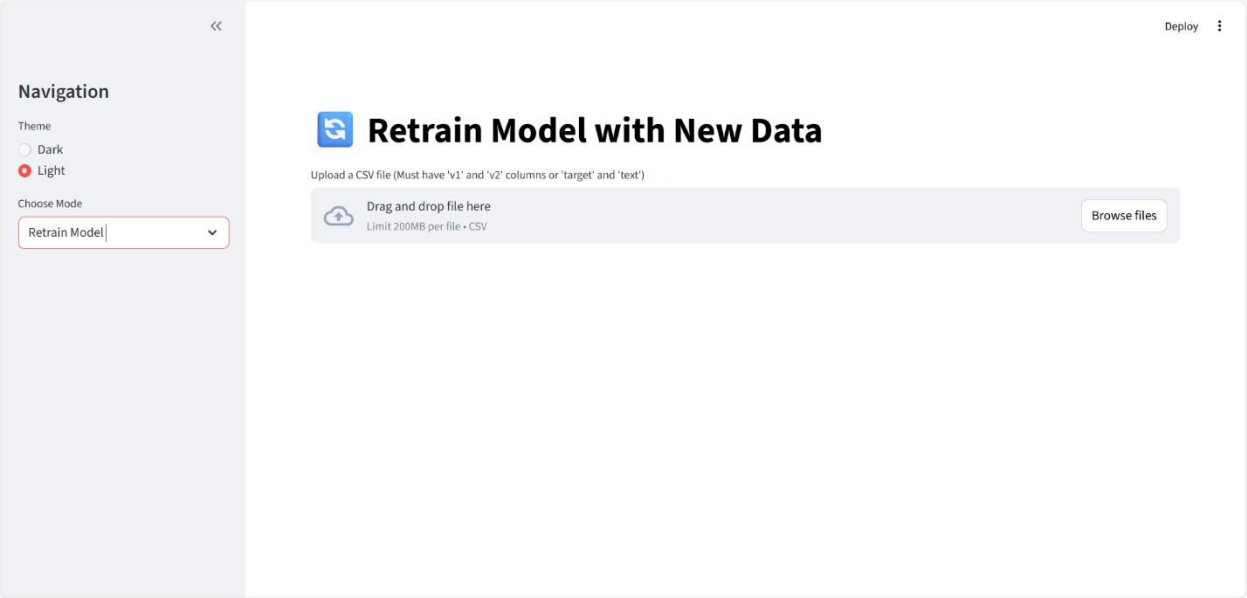


Figure 6.1

6.2 HISTORY AND ANALYTICS :



RETRAIN MODEL WITH NEW DATA



SPAM OR HAAM EMAIL DETECTION