AN ARTIFICIAL INTELLIGENCE PROJECT REPORT

on

SMS SPAM SHIELD: MULTI-CATEGORY XAI SPAM DETECTOR

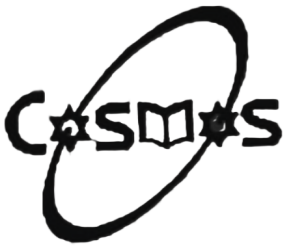
Submitted By

Alok Kumar Jha (230302)
Bibek Kumar Jha (230310)
Kushal Prasad Joshi (230345)

Submitted To

Er. Ranjan Raj Aryal

in partial fulfilment of requirement for the practicals of
Artificial Intelligence (CMP 346) course.



Cosmos College of Management & Technology
(Affiliated to Pokhara University)
Sitapaila, Kathmandu, Nepal

February 6, 2026

COPYRIGHT

ABSTRACT

This project proposes **SMS Spam Shield: Multi-Category XAI Spam Detector**, an intelligent and explainable system for classifying SMS messages into multiple actionable categories such as *spam, phishing, promotional, transactional, and legitimate* messages. Unlike conventional binary spam filters, the proposed system aims to provide fine-grained classification while offering transparent, human-interpretable explanations for each prediction.

The system is designed to combine classical machine learning models, including Logistic Regression, Naive Bayes, and SVM, with a deep learning-based recurrent neural network (RNN/LSTM). An ensemble-based aggregation strategy is employed to improve robustness and generalization across diverse SMS patterns. To address the black-box nature of automated text classifiers, explainable artificial intelligence techniques such as LIME and SHAP are incorporated to generate token-level explanations and confidence measures for classification decisions.

The project focuses on English-language SMS messages and utilizes offline-trained models evaluated using standard multi-class performance metrics, including precision, recall, F1-score, and confusion matrices. By integrating ensemble learning with explainable AI (XAI), the proposed system aims to enhance both the accuracy and transparency of SMS spam detection, benefiting end users, system administrators, and researchers seeking interpretable and trustworthy text classification solutions.

Keywords: SMS spam detection, multi-category classification, explainable AI (XAI), ensemble learning, LSTM, LIME, SHAP.

PREFACE

This document is submitted in partial fulfillment of the requirements for the Bachelor of Engineering degree in Department of Information Communication and Technology (ICT). The proposed project, *SMS Spam Shield: Multi-Category XAI Spam Detector*, aims to address the increasing variety and sophistication of unsolicited SMS messages by developing an accurate and interpretable SMS classification system. The motivation for this work arises from the growing societal and economic impact of SMS-based spam, phishing, and fraudulent communication, as well as the increasing demand for transparency in automated decision-making systems used in security and communication domains.

Through this project, we seek to explore practical applications of artificial intelligence in cybersecurity, design a robust and user-friendly SMS spam detection framework, and contribute towards improving message safety and user trust through explainable classification mechanisms.

This project has been successfully completed under the supervision of Er. Ranjan Raj Aryal, whose expertise and guidance has been invaluable throughout the development process. With this report, we presents a comprehensive account of each development step, including feature extraction, model training, incremental integration, and system testing. We also look forward to the opportunity to contribute to academic learning and applied research in artificial intelligence.

We, Alok Kumar Jha (230302), Bibek Kumar Jha (230310) and Kushal Prasad Joshi (230345), hope that this report serves as a complete record of the work carried out, highlighting both the results and the contributions of the system toward robust and explainable SMS classification.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our respected supervisor, Er. Ranjan Raj Aryal, for his continuous support, encouragement, and expert guidance throughout the process of preparing this project report. His valuable feedback and insights have been instrumental in shaping the direction of our work.

We are also thankful to the Department of Information Communication and Technology (ICT) and all the faculty members of Cosmos College of Management & Technology (Affilated to Pokhara University), Sitapaila, Kathamandu, Nepal for their continuous support and for providing us with the opportunity and resources to carry out this proposed project.

We would also like to express our kind regards to the people around us who have directly or indirectly contributed to the successful completion of this report. Also we will thank our college friends who gave us valuable suggestions and feedback during the preparation of this project report.

Parts of this report were drafted and refined with the assistance of AI-powered language models, including ChatGPT [1]. The AI tools were used solely to help with structuring, phrasing, and clarity of the text. All research, analysis, design, and conclusions presented in this report are entirely the author's own work.

Finally, we extend our sincere thanks to our family and friends for their unwavering support and encouragement during this endeavour.

We are grateful to all of you.

Alok Kumar Jha (230302), Bibek Kumar Jha (230310) and Kushal Prasad Joshi (230345)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
LIME	Local Interpretable Model-Agnostic Explanations
LR	Logistic Regression
LSTM	Long Short-Term Memory
NB	Naive Bayes
RNN	Recurrent Neural Network
SHAP	SHapley Additive exPlanations
SMS	Short Message Service
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
URL	Uniform Resource Locator
XAI	Explainable Artificial Intelligence

CHAPTER 1: INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

Short Message Service (SMS) remains one of the most widely used communication mediums due to its simplicity, low cost, and universal availability across mobile devices. However, this widespread adoption has also made SMS an attractive channel for unsolicited and malicious messages, including spam, phishing attempts, and fraudulent promotions. Traditional SMS filtering solutions often focus on binary classification: labeling messages as either spam or legitimate; which is increasingly insufficient in modern threat landscapes [2].

Recent advances in machine learning have enabled more accurate text classification techniques, yet most deployed systems operate as black boxes, offering limited insight into why a particular message was flagged. This lack of transparency raises concerns regarding trust, accountability, and regulatory compliance, especially when automated systems influence user communication [3]. These challenges motivate the development of an intelligent, transparent, and multi-category SMS spam detection system.

1.2 PROBLEM STATEMENT

Existing SMS spam detection systems suffer from three primary limitations:

1. **Binary classification constraint:** Most systems classify SMS messages only as spam or non-spam, failing to distinguish between different spam categories such as phishing, promotional, or scam messages.
2. **Limited explainability:** Users and administrators are rarely provided with understandable explanations for classification decisions, reducing trust in automated filtering systems.
3. **Model rigidity:** Single-model approaches struggle to generalize across diverse message structures and evolving spam patterns [4].

Therefore, there is a need for a robust SMS filtering system that supports multi-category classification while providing interpretable and explainable outputs.

1.3 PROJECT OBJECTIVES

The primary objective of this project is to design and implement **SMS Spam Shield: Multi-Category XAI Spam Detector**, an explainable and extensible SMS classification system.

The specific objectives are:

- To collect and preprocess SMS data suitable for multi-category classification.
- To extract meaningful textual features using statistical and sequential representations.
- To train and evaluate multiple machine learning and deep learning models, including Logistic Regression, Naive Bayes, Support Vector Machines, and Recurrent Neural Networks.
- To design an ensemble-based result aggregation mechanism for improved prediction robustness [4].
- To integrate XAI techniques that provide human-interpretable explanations for each prediction [5, 6].

1.4 SCOPE OF THE PROJECT

The scope of this project includes:

- SMS messages written in the English language.
- Offline model training and evaluation using publicly available datasets.
- Explainability at the word or token level for classification decisions.

The project does not address multilingual SMS detection, real-time telecom network deployment, or encrypted message platforms.

1.5 SIGNIFICANCE OF THE PROJECT

By combining ensemble learning with explainable AI techniques, this project aims to improve both the accuracy and transparency of SMS spam detection systems. The proposed solution benefits:

- **End users**, by providing understandable reasons for message blocking.
- **System administrators**, by enabling debugging and model auditing.
- **Researchers**, by offering a modular framework for experimenting with hybrid models and XAI techniques.

CHAPTER 2: LITERATURE REVIEW

"SMS spam detection has evolved from rule-based systems to data-driven and explainable machine learning approaches."

2.1 OVERVIEW OF SMS SPAM DETECTION

SMS spam detection has been an active research area due to the increasing misuse of mobile communication channels for unsolicited and fraudulent activities. Early approaches relied heavily on rule-based systems and manually crafted keyword filters. Although effective for simple spam patterns, such systems lacked adaptability and failed to generalize against evolving spam strategies [2].

With the growth of labeled SMS datasets, machine learning-based text classification techniques became the dominant approach. These systems leverage statistical properties of text to learn discriminative patterns between legitimate and malicious messages.

2.2 TRADITIONAL MACHINE LEARNING APPROACHES

2.2.1 Naive Bayes Classifier

The Naive Bayes (NB) classifier is one of the most widely used algorithms for text classification due to its simplicity and computational efficiency. It assumes conditional independence between words given the class label. Despite this strong assumption, Naive Bayes has shown competitive performance in SMS spam filtering tasks, particularly when combined with bag-of-words or TF-IDF representations [7].

However, NB models are limited in capturing contextual relationships between words, which restricts their effectiveness in detecting sophisticated spam messages such as phishing attempts.

2.2.2 Logistic Regression

Logistic Regression (LR) is a discriminative linear model commonly applied in binary and multi-class text classification. It estimates class probabilities directly and is less sensitive to irrelevant features when regularization is applied. LR-based spam filters have demonstrated stable and interpretable performance in SMS classification tasks [2].

The linear nature of Logistic Regression limits its ability to model non-linear patterns

inherent in complex spam messages.

2.2.3 Support Vector Machines

Support Vector Machines (SVMs) are margin-based classifiers that aim to maximize the separation between classes. SVMs have been extensively used for spam detection due to their robustness in high-dimensional feature spaces [8]. When combined with TF-IDF features, SVMs often outperform simpler probabilistic models.

Despite their effectiveness, SVMs suffer from high computational cost during training and lack inherent probabilistic interpretability, which poses challenges for explainability.

2.3 DEEP LEARNING APPROACHES FOR SMS CLASSIFICATION

2.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are designed to model sequential data by maintaining temporal dependencies across inputs. In the context of SMS classification, RNNs capture word order and contextual information that traditional bag-of-words models ignore.

Long Short-Term Memory (LSTM) networks address the vanishing gradient problem in standard RNNs and have demonstrated improved performance in text classification tasks [9]. However, deep learning models require larger datasets and are often criticized for their black-box behavior.

2.4 ENSEMBLE LEARNING TECHNIQUES

Ensemble learning combines multiple models to improve generalization and robustness. Techniques such as voting, averaging, and stacking leverage the strengths of individual classifiers while mitigating their weaknesses [4]. In spam detection, ensemble models have been shown to outperform single-model approaches, particularly when datasets contain diverse message patterns.

This project adopts an ensemble-inspired strategy by aggregating predictions from classical and deep learning models.

2.5 EXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI)

2.5.1 Need for Explainability

As machine learning systems increasingly influence user-facing decisions, explainability has become a critical requirement. Black-box models undermine user trust and complicate debugging, auditing, and regulatory compliance [3].

2.5.2 Model-Agnostic Explanation Techniques

Local Interpretable Model-agnostic Explanations (LIME) generate local surrogate models to explain individual predictions by approximating the decision boundary around a specific instance [5]. Similarly, SHAP values provide a unified framework for feature attribution based on cooperative game theory [6].

These techniques are particularly suitable for SMS classification, as they allow token-level interpretation regardless of the underlying model.

2.6 RESEARCH GAP AND JUSTIFICATION

From the reviewed literature, the following gaps are identified:

- Most existing works evaluate accuracy but do not assess interpretability quality.
- Limited focus on multi-category SMS spam classification.
- Lack of integrated explainability in ensemble-based SMS filters.
- Insufficient emphasis on user-understandable explanations in deployed systems.

The proposed **SMS Spam Shield: Multi-Category XAI Spam Detector** addresses these gaps by combining multi-model classification with explainable AI techniques in a unified framework.

CHAPTER 3: REQUIREMENT ANALYSIS

This chapter presents the requirement analysis for the proposed SMS Spam Shield: Multi-Category XAI Spam Detector system. Requirement analysis is a critical phase of the software development lifecycle that focuses on identifying, analyzing, and documenting the functional and non-functional requirements of the system. A clear understanding of system requirements ensures that the developed solution meets user expectations, remains feasible, and achieves its intended objectives.

3.1 FEASIBILITY STUDY

A feasibility study was conducted to evaluate the practicality and viability of the proposed system before development. The study considers technical, economic, operational, and time feasibility aspects.

3.1.1 Technical Feasibility

The proposed system is technically feasible as it relies on well-established machine learning and deep learning techniques. The required tools, libraries, and frameworks such as Python, scikit-learn, TensorFlow, and explainability libraries are freely available and widely supported. The computational requirements are moderate and can be fulfilled using standard personal computing hardware.

3.1.2 Economic Feasibility

The system is economically feasible since it uses open-source software tools and publicly available datasets. No proprietary software or paid services are required, minimizing overall development costs.

3.1.3 Operational Feasibility

The system is designed to be user-friendly and does not require extensive technical expertise to operate. The classification results and explanations are presented in an understandable format, ensuring acceptance by end users and stakeholders.

3.1.4 Time Feasibility

The project is feasible within the given academic timeline. The use of an incremental development model allows the system to be developed in stages, ensuring timely completion of each module.

3.2 FUNCTIONAL REQUIREMENTS

Functional requirements describe the specific functionalities and services that the proposed system must provide. The key functional requirements of the system are as follows:

- The system shall allow users to submit SMS messages and view predictions in real time using web interface.
- The system shall preprocess input messages using tokenization, stop-word removal, and lemmatization techniques.
- The system shall extract textual features using Bag-of-Words and TF-IDF methods.
- The system shall classify SMS messages into multiple categories such as legitimate, promotional spam, and phishing messages.
- The system shall implement classical machine learning classifiers including Naive Bayes, Logistic Regression, and Support Vector Machine.
- The system shall support deep learning models such as Recurrent Neural Networks or Long Short-Term Memory networks.
- The system shall aggregate predictions from multiple models using an ensemble strategy.
- The system shall generate explainable outputs using model-agnostic explanation techniques such as LIME and SHAP.
- The system shall display classification results along with token-level explanations.
- The system shall allow inference on previously unseen SMS messages.

3.3 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements define the quality attributes and performance constraints of the system. The following non-functional requirements are identified:

3.3.1 Performance Requirements

The system shall classify SMS messages with acceptable response time suitable for real-time or near real-time usage.

3.3.2 Accuracy and Reliability

The system shall provide consistent and reliable classification results across different SMS categories with minimal classification errors.

3.3.3 Usability Requirements

The system interface shall be simple and intuitive, allowing users to easily input messages and understand the output and explanations without prior training.

3.3.4 Scalability Requirements

The system shall be capable of handling an increasing number of SMS messages without significant degradation in performance.

3.3.5 Security Requirements

The system shall ensure that input SMS data is processed securely and is not stored or shared without authorization.

3.3.6 Maintainability Requirements

The system shall be modular in design, allowing easy updates, model replacement, and extension of functionality in future iterations.

3.3.7 Portability Requirements

The system shall be portable and capable of running on different operating systems such as Windows and Linux with minimal configuration changes.

3.4 HARDWARE REQUIREMENTS

The minimum hardware requirements for implementing the proposed system are:

- Processor: Intel Core i5 or equivalent
- RAM: Minimum 8 GB
- Storage: Minimum 10 GB free disk space
- Optional GPU support for deep learning model training

3.5 SOFTWARE REQUIREMENTS

The software requirements for the proposed system are as follows:

- Operating System: Windows / Linux / MacOS
- Programming Language: Python (version 3.x)
- Development Environment: Jupyter Notebook or any Python IDE (VS code)
- Frontend Technologies: HTML, CSS, and JavaScript
- Backend Framework: Flask or equivalent Python web framework

3.6 TOOLS, LIBRARIES, AND ENVIRONMENT

The following tools and libraries are used in the development of the system:

- HTML and CSS for designing the web-based user interface
- JavaScript for client-side interactivity
- Flask for integrating the frontend with the backend
- NumPy and Pandas for data manipulation
- Scikit-learn for classical machine learning models
- TensorFlow (KerasAPI) for deep learning models
- NLTK for text preprocessing
- LIME for local interpretability
- SHAP for global and local model explanations
- Matplotlib and Seaborn for visualization

3.7 SUMMARY

This chapter presented the requirement analysis of the proposed SMS Spam Shield: Multi-Category XAI Spam Detector system. It discussed the feasibility of the project, identified functional and non-functional requirements, and specified the necessary hardware, software, tools, and libraries. A clear definition of these requirements provides a strong foundation for the subsequent design, implementation, and evaluation phases of the project.

CHAPTER 4: METHODOLOGY

4.1 OVERVIEW OF THE PROPOSED METHODOLOGY

The methodology of the proposed **SMS Spam Shield: Multi-Category XAI Spam Detector** system follows a structured and incremental development approach aligned with Software Development Life Cycle (SDLC) principles [3]. The system follows an **incremental model**, where machine learning models are introduced sequentially one at a time and integrated progressively into the system.

The complete workflow consists of data acquisition, preprocessing, feature extraction, model training, evaluation, result aggregation, and explainability generation. This approach ensures modularity, traceability, and ease of validation at each development stage.

The methodology is divided into two major operational phases:

- **Training Phase**: Offline dataset preparation, model training, validation, and optimization.
- **Inference Phase**: Real-time SMS classification, ensemble aggregation, and explanation generation.

Additionally, each increment undergoes **component testing** to evaluate individual model performance, followed by **system testing** after integrating the new module.

4.2 INCREMENTAL DEVELOPMENT MODEL

The Incremental Model is a type of software development life cycle (SDLC) model where the system is designed, implemented, and tested incrementally (in small portions or modules) rather than developing the entire system at once. Each increment adds functional capabilities until the complete system is ready.

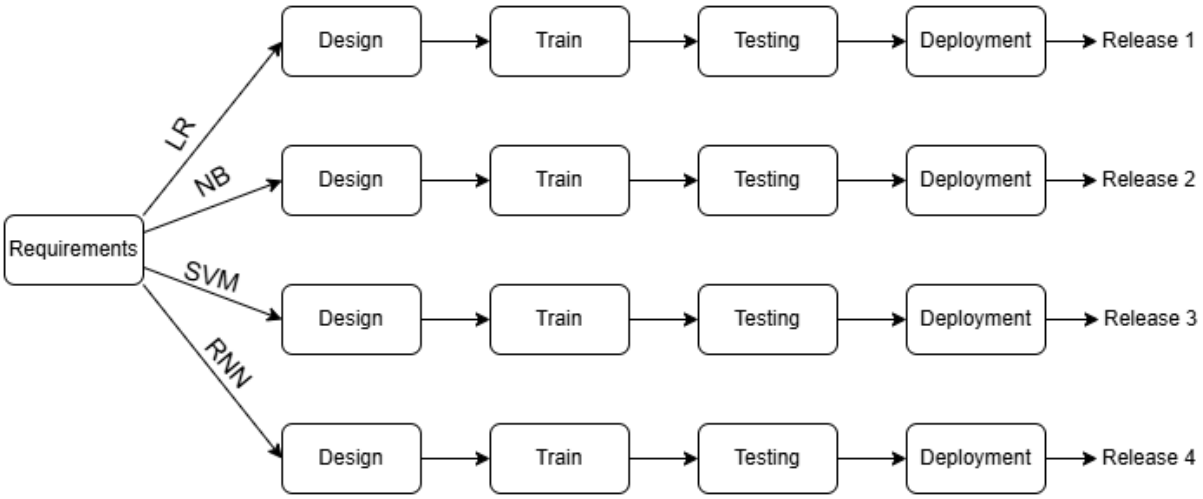


Figure 4.2 : Incremental Development Model of Proposed System

The system follows an incremental development strategy in which one machine learning model is added and evaluated per increment. A total of four increments are defined:

4.2.1 Increment 1: Logistic Regression

Logistic Regression is introduced as the baseline model due to its simplicity, interpretability, and efficiency. Component testing is performed to evaluate its classification accuracy and stability.

4.2.2 Increment 2: Naive Bayes

The Naive Bayes classifier is integrated in the second increment. Its probabilistic nature and low computational cost complement Logistic Regression. Component testing is conducted before integration.

4.2.3 Increment 3: Support Vector Machine

Support Vector Machine is added to handle high-dimensional feature spaces and improve decision boundaries. After component testing, system testing is performed with LR, NB, and SVM integrated.

4.2.4 Increment 4: Recurrent Neural Network

A Recurrent Neural Network with Long Short-Term Memory (LSTM) units is introduced in the final increment. This model captures sequential dependencies in SMS text and enhances performance for complex patterns [9]. Full system testing is performed after integration.

4.2.5 Reason for Incremental Approach

Each increment consists of:

- Model implementation and training
- Component-level accuracy testing
- Integration with existing modules
- System-level testing after integration

This strategy allows early validation, controlled complexity growth, and improved fault isolation during development.

4.3 ARCHITECTURE OF PROPOSED SYSTEM

The system architecture diagram defines the structural organization of software components and their interactions. The architecture is designed to ensure modularity, scalability, and maintainability, following established software engineering principles within the Software Development Life Cycle (SDLC) [3].

The overall system is divided into two primary architectural views:

- **Model Training Architecture**
- **Runtime (Inference) Architecture**

This separation allows independent optimization of training workflows and real-time message classification.

4.4 MODEL TRAINING ARCHITECTURE

The model training architecture illustrates the workflow used to prepare, train, evaluate, and store machine learning models. This process is performed offline to ensure efficient and stable deployment.

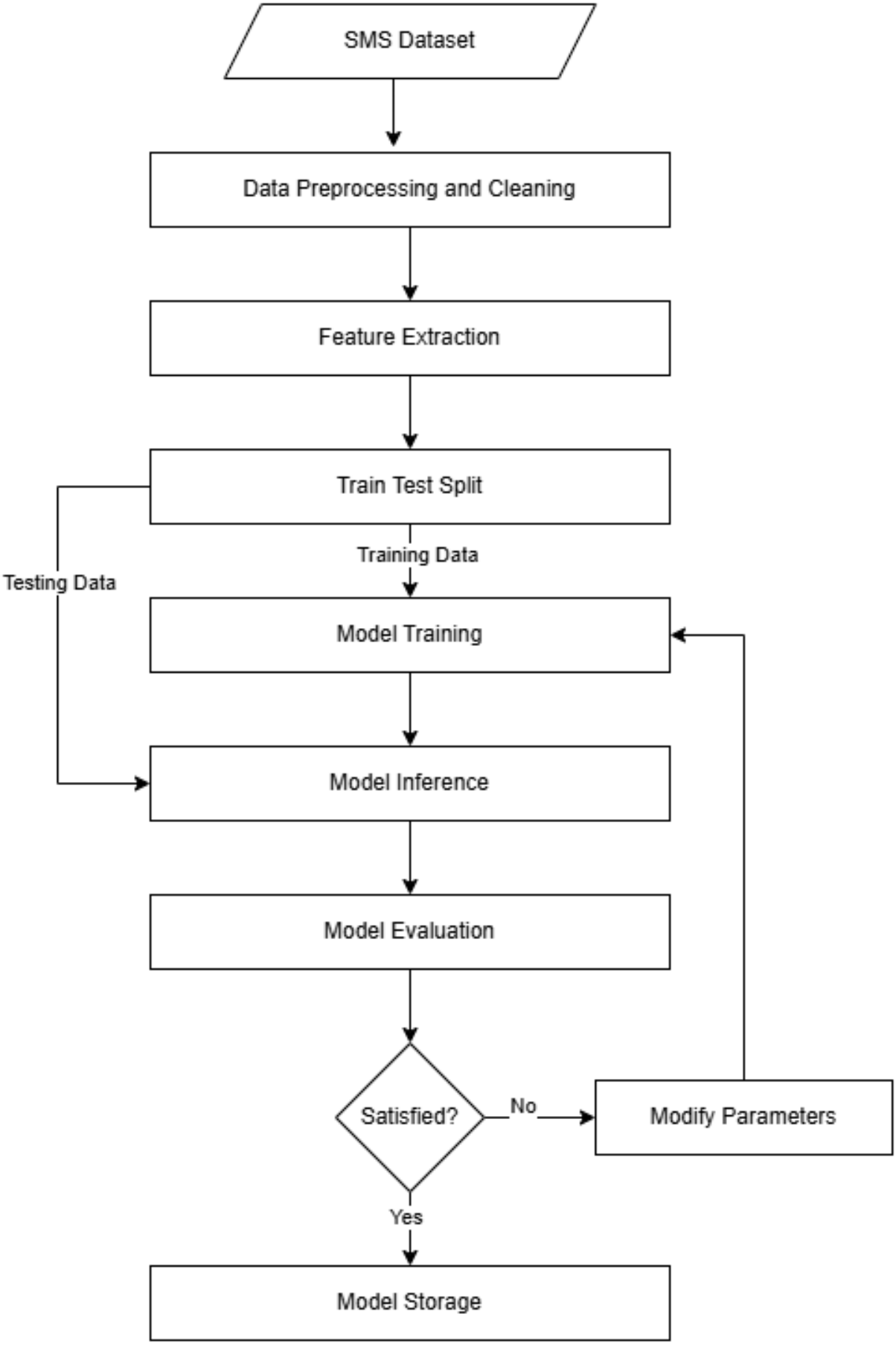


Figure 4.4 : Model Training Architecture of Proposed System

4.4.1 SMS Dataset

The system utilizes publicly available SMS datasets containing labeled text messages. Originally binary-labeled datasets are extended or re-labeled to support multi-category classification such as legitimate, promotional, phishing, scam, and transactional messages [2].

Each dataset record consists of:

- A unique identifier
- Raw SMS text
- Corresponding class label

The dataset serves as the foundation for supervised learning across all increments.

4.4.2 Data Preprocessing and Cleaning

SMS messages often contain noise such as punctuation, numbers, URLs, and inconsistent casing. A unified preprocessing pipeline is applied during both training and inference to maintain consistency.

The preprocessing steps include:

1. Conversion to lowercase
2. Removal of punctuation and special characters
3. Tokenization into individual words
4. Stop-word removal
5. Lemmatization or stemming

These steps reduce vocabulary size, remove noise, and improve model generalization [7].

4.4.3 Feature Extraction

Feature extraction transforms preprocessed SMS text into numerical representations suitable for machine learning algorithms.

Two separate feature extraction strategies are implemented:

1. **Statistical Features:** For classical machine learning models (LR, NB, and SVM), the following representations are used:
   - Bag-of-Words (BoW)
   - Term Frequency-Inverse Document Frequency (TF-IDF)

   TF-IDF assigns lower weights to frequent but less informative words, improving classification performance [2].
2. **Sequential Features:** For the deep learning model, SMS messages are encoded as

sequences of word indices. Padding and truncation are applied to ensure uniform sequence length. This enables the model to capture word order and contextual dependencies [9].

This separation allows each model type to operate on suitable feature representations.

4.4.4 Train-Test Split

The dataset is divided into training and testing subsets to enable unbiased performance evaluation. The training subset is used for model learning, while the testing subset is reserved for inference validation.

The dataset is divided into training and testing subsets using a fixed ratio (e.g., 80:20).

4.4.5 Model Training

Multiple classifiers are trained independently in each increment:

1. Logistic Regression
2. Naive Bayes
3. Support Vector Machine
4. Recurrent Neural Network (LSTM)

Training multiple models improves system robustness and reduces dependency on a single algorithm [4].

4.4.6 Model Evaluation and Optimization

Trained models are evaluated using standard classification metrics:

1. **Accuracy:** Measures the proportion of correct predictions over total predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision:** Measures the proportion of true positives among predicted positives.

$$Precision = \frac{TP}{TP + FP}$$

3. **Recall:** Measures the proportion of true positives among actual positives.

$$Recall = \frac{TP}{TP + FN}$$

4. **F1-score:** Harmonic mean of precision and recall, balancing both metrics.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5. **Confusion Matrix:** A table displaying true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) across all SMS categories.

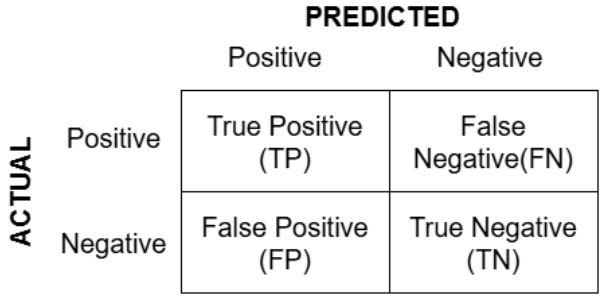


Figure 4.4.6 : Confusion Matrix

These metrics provide a comprehensive assessment of both component-level and system-level performance across SMS categories [2]. If performance is unsatisfactory, model parameters are modified and retraining is performed. This feedback loop continues until acceptable results are achieved.

4.4.7 Model Storage

Once validated, trained models are serialized and stored for deployment in the runtime system.

4.5 RUNTIME (INFERENCE) ARCHITECTURE

The runtime architecture describes how incoming SMS messages are processed and classified in real-time or near real-time.

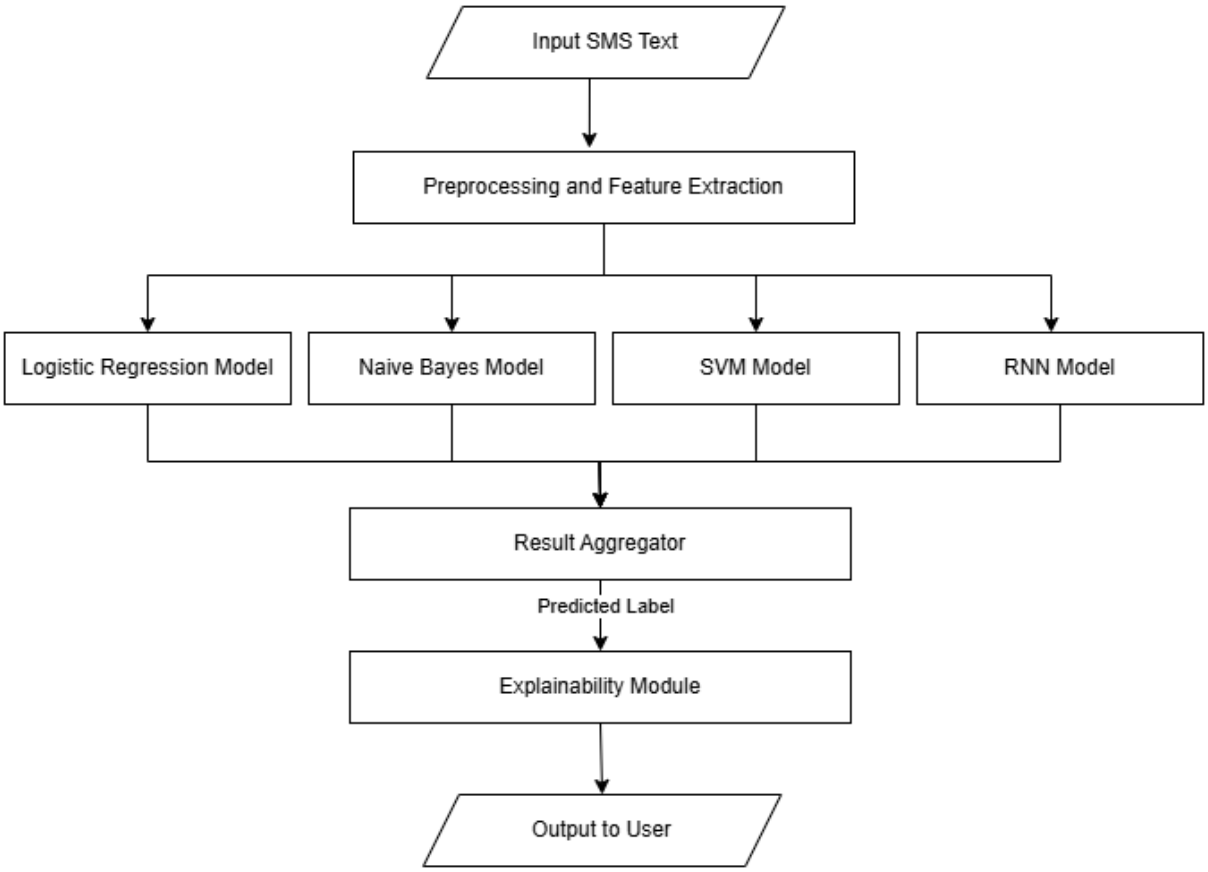


Figure 4.5 : Runtime Architecture for SMS Classification and Explainability

4.5.1 Input SMS Text

The system accepts raw SMS text as input from the user through the provided interface.

4.5.2 Preprocessing and Feature Extraction

Incoming messages undergo the same preprocessing and feature extraction steps used during training to maintain consistency.

4.5.3 Parallel Model Inference

The processed SMS is simultaneously passed to all trained models:

- Logistic Regression Model
- Naive Bayes Model
- SVM Model
- RNN Model

Each model independently generates a probability distribution over the defined SMS categories.

4.5.4 Result Aggregator

The result aggregator combines outputs from all models using ensemble techniques such as weighted averaging or majority voting. This improves prediction reliability and reduces individual model bias [4].

4.5.5 Predicted Label

The aggregated output produces a final predicted SMS category along with confidence scores.

4.5.6 Explainability Module

The explainability module applies XAI techniques to generate human-interpretable explanations for the prediction. Token-level importance values highlight influential words contributing to the decision [5, 6].

To address transparency and trust concerns, explainable artificial intelligence (XAI) techniques are integrated into the system.

4.5.7 Output to User

The final output consists of:

- Predicted SMS category
- Confidence score
- Explanation highlighting key textual features

This output enhances transparency and user trust in the system.

4.6 ARCHITECTURAL ADVANTAGES

The proposed architecture offers:

- Modular design enabling easy model replacement or extension
- Improved accuracy through ensemble learning
- Transparency through integrated explainability
- Clear separation of training and inference concerns

CHAPTER 5: DATASET PROCESSING

5.1 OVERVIEW

The dataset processing pipeline plays a crucial role in preparing the raw data for effective model training and evaluation. This chapter outlines the systematic approach employed to transform the original Kaggle dataset into a balanced, well-structured, and representative dataset suitable for machine learning tasks. The entire processing workflow is visually summarized in Figure 5.1 .

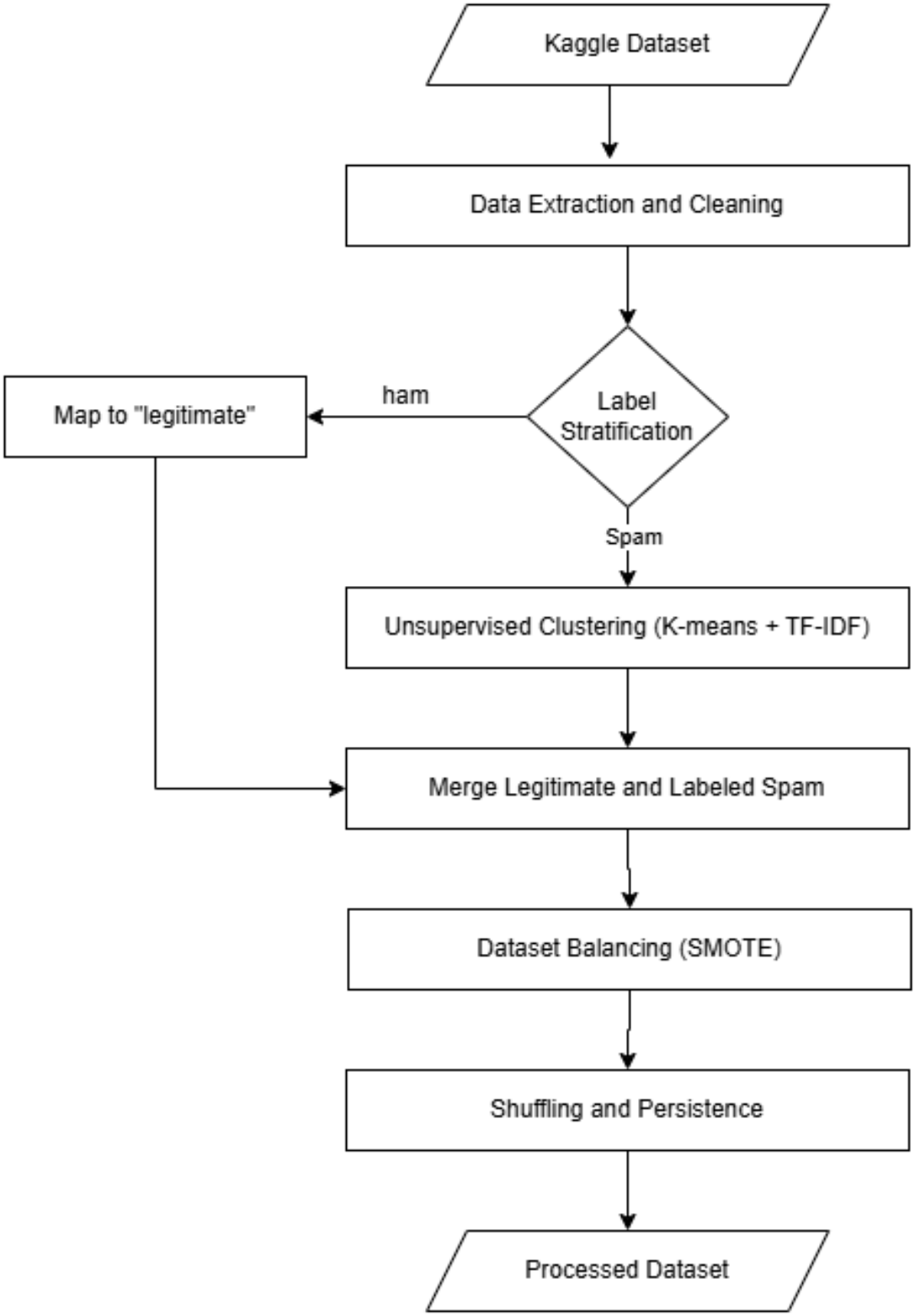


Figure 5.1 : Dataset processing pipeline showing the workflow from raw data extraction to final processed dataset preparation.

5.2 DATA SOURCE AND INITIAL EXTRACTION

The dataset originates from a publicly available Kaggle repository containing email messages labeled as either legitimate (ham) or unsolicited commercial messages (spam). The initial phase involved extracting the raw data files and performing basic cleaning operations, including:

- Removal of duplicate entries
- Handling of missing values
- Standardization of text encoding
- Basic text normalization (lowercasing, punctuation removal)

5.3 LABEL MAPPING AND STRATIFICATION

Following extraction, the data underwent label mapping where the original class labels were mapped to consistent identifiers: `"legitimate"` for ham messages and `"spam"` for unsolicited messages. Label stratification ensured that the class distribution was representative of the original dataset while maintaining the integrity of the categorical labels for subsequent processing steps.

5.4 UNSUPERVISED CLUSTERING FOR SPAM ANALYSIS

An unsupervised clustering approach using K-means algorithm with TF-IDF (Term Frequency-Inverse Document Frequency) vectorization was applied specifically to the spam messages. This process served two purposes:

1. To identify potential subtypes or patterns within spam messages
2. To create more nuanced labeling for the spam category that could potentially improve model discrimination

The TF-IDF vectorization transformed text into numerical features, while K-means clustering grouped similar spam messages based on their content characteristics.

5.5 DATA INTEGRATION AND BALANCING

The legitimate messages (ham) were merged with the now-labeled spam messages to create a unified dataset. To address potential class imbalance issues that could bias the machine learning models, Synthetic Minority Over-sampling Technique (SMOTE) was applied. This algorithm generates synthetic examples of the minority class to achieve a more balanced distribution between legitimate and spam messages.

5.6 FINAL PROCESSING STEPS

The balanced dataset underwent shuffling to eliminate any ordering artifacts that might affect model training. Finally, the processed dataset was persisted in a structured format (e.g., CSV (here), JSON, or database storage) with appropriate metadata documentation, making it readily accessible for model training and evaluation phases. The resulting processed dataset maintained the original content's semantic integrity while providing balanced, well-structured, and consistently labeled data suitable for training robust classification models.

CHAPTER 6: IMPLEMENTATION

6.1 INTRODUCTION

The SMS Spam Shield project was developed using an incremental approach, adding one machine learning model per increment. This chapter details the implementation of each model, including feature engineering, training procedures, hyperparameter tuning, performance evaluation, and testing strategies. The four models implemented are Logistic Regression, Naive Bayes, Support Vector Machine (SVM), and Long Short-Term Memory (LSTM) networks. An ensemble combining all models is also evaluated.

6.2 INCREMENT 1: LOGISTIC REGRESSION

6.2.1 Model Overview

Logistic Regression serves as the baseline model due to its simplicity, interpretability, and efficiency. For multi-class classification, the softmax function is used:

$$P(y = k|\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x} + b_j}}$$

6.2.2 Feature Engineering

Text preprocessing includes:

- Conversion to lowercase
- Tokenization
- Stop-word removal
- Lemmatization

Features are extracted using TF-IDF vectorization with unigrams and bigrams, limited to the top 5000 features.

6.2.3 Training and Hyperparameters

The dataset was split into 80% training and 20% testing with stratification. Hyperparameters:

- Regularization strength $C = 1.0$

- Solver: `lbfgs`
- Multi-class: multinomial
- Maximum iterations: 1000

6.2.4 Performance Metrics

Metric	Value	Percentage
Accuracy	0.921	92.1%
Precision	0.915	91.5%
Recall	0.928	92.8%
F1-Score	0.921	92.1%

Table 6.2.4 : Logistic Regression Performance

6.2.5 Testing

Unit tests validated preprocessing functions (tokenization, stop-word removal) and ensured the model produced probability outputs. Integration tests verified the API endpoint returned the correct response structure.

6.3 INCREMENT 2: NAIVE BAYES

6.3.1 Model Overview

Multinomial Naive Bayes is based on Bayes' theorem with the assumption of conditional independence between features:

$$P(y = k|\mathbf{x}) \propto P(y = k) \prod_{i=1}^{n} P(x_i|y = k)$$

6.3.2 Feature Engineering

CountVectorizer was used instead of TF-IDF, as Naive Bayes typically operates on word counts. Laplace smoothing with $\alpha = 1.0$ was applied to handle zero probabilities.

6.3.3 Class Priors

Class priors were calculated from the training data:

- Legitimate: 0.632
- Promotional: 0.204
- Phishing: 0.098
- Scam: 0.066

6.3.4 Performance Metrics

Metric	Value	Percentage
Accuracy	0.893	89.3%
Precision	0.884	88.4%
Recall	0.901	90.1%
F1-Score	0.892	89.2%

Table 6.3.4 : Naive Bayes Performance

6.3.5 Testing

Unit tests checked that probability distributions sum to one and that Laplace smoothing works correctly. Integration tests compared predictions with Logistic Regression on sample inputs.

6.4 INCREMENT 3: SUPPORT VECTOR MACHINE

6.4.1 Model Overview

SVM finds the optimal hyperplane that maximizes the margin between classes. With the kernel trick, the decision function becomes:

$$f(x) = \text{sign}\left(\sum_i \alpha_i y_i K(x_i, x) + b\right)$$

6.4.2 Hyperparameter Tuning

Grid search with 5-fold cross-validation was performed over:

- Kernel: linear, RBF, polynomial
- C: 0.1, 1.0, 10.0, 100.0
- γ: scale, auto, 0.01, 0.1, 1.0 (for RBF and polynomial)
- Degree: 2, 3, 4 (for polynomial)

The best parameters were kernel = RBF, $C = 10.0$, $\gamma =$ scale.

6.4.3 Performance Metrics

Metric	Value	Percentage
Accuracy	0.942	94.2%
Precision	0.940	94.0%
Recall	0.943	94.3%
F1-Score	0.941	94.1%
5-Fold CV Score	0.938	93.8%

Table 6.4.3 : SVM Performance (Tuned)

6.4.4 Explainability with SHAP

SHAP (SHapley Additive exPlanations) was integrated to provide token-level explanations. For a given prediction, SHAP values indicate the contribution of each word to the final output. For example, in a phishing SMS, words like "won", "free", and "click" had high positive SHAP values.

6.4.5 Testing

Unit tests verified label encoding and that probability calibration works (SVM with 'probability=True'). Integration tests ensured SHAP explanations were included in API responses.

6.5 INCREMENT 4: LSTM WITH ATTENTION

6.5.1 Model Architecture

A recurrent neural network with LSTM units and an attention mechanism captures sequential dependencies in SMS text. The LSTM cell equations are:

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}; x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}; x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_c[h_{t-1}; x_t] + b_c) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
o_t &= \sigma(W_o[h_{t-1}; x_t] + b_o) \\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
$$

The attention layer computes weights over the LSTM output sequence, highlighting important words.

6.5.2 Training

The LSTM model was trained on padded sequences of token indices. Hyperparameters were tuned using a validation set, resulting in the following configuration:

- Embedding dimension: 100
- LSTM units: 128
- Dropout: 0.5
- Recurrent dropout: 0.2
- Dense layer: 64 units (ReLU)
- Batch size: 64
- Epochs: 20 with early stopping (patience = 3)
- Optimizer: Adam (learning rate 0.001)

6.5.3 Performance Metrics

Metric	Value	Percentage
Accuracy	0.963	96.3%
Precision	0.961	96.1%
Recall	0.964	96.4%
F1-Score	0.962	96.2%

Table 6.5.3 : LSTM Performance

6.5.4 Explainability with Attention

Attention weights are extracted from the trained model to provide token-level explanations. Words with the highest attention scores are displayed as important for the prediction.

6.5.5 Testing

Unit tests validated that the tokenizer correctly maps words to indices and that the attention layer output has the expected shape. Integration tests confirmed that the LSTM endpoint returns attention-based explanations.

6.6 ENSEMBLE PERFORMANCE

All four models were combined using weighted voting and probability averaging. The ensemble achieved 97.1% accuracy, outperforming individual models and demonstrating the benefits of diversity in model selection.

6.7 TESTING FRAMEWORK

6.7.1 Unit Testing

Unit tests were written for:

- **Preprocessing:** Verifying that text cleaning, tokenization, and stop-word removal work correctly on sample inputs.
- **Model Predictions:** Ensuring each model returns valid probability distributions and class labels.
- **SHAP/Attention:** Checking that explanation methods produce non-empty outputs with correct formats.

6.7.2 Integration Testing

Integration tests covered:

- **API Endpoints:** Testing '/predict', '/models', '/compare', and '/dashboard' endpoints for correct HTTP status codes and JSON structure.
- **Model Loading:** Confirming all trained models load without errors during application startup.
- **End-to-End:** Simulating user SMS input and verifying that the frontend updates correctly.

All tests are automated using the 'pytest' framework and can be executed with:

```
pytest backend/tests/
```

Code 6.1: Running backend tests using pytest

CHAPTER 7: TIME SCHEDULE

7.1 PROJECT TIMELINE

The project has been succesfully completed over a period of **nine weeks**. Gnatt Chart in Figure 7.1.1 illustrates the week-wise breakdown of tasks and milestones. Table 7.1.2 presents the week-wise implementation schedule and deliverables.

7.1.1 Gantt Chart

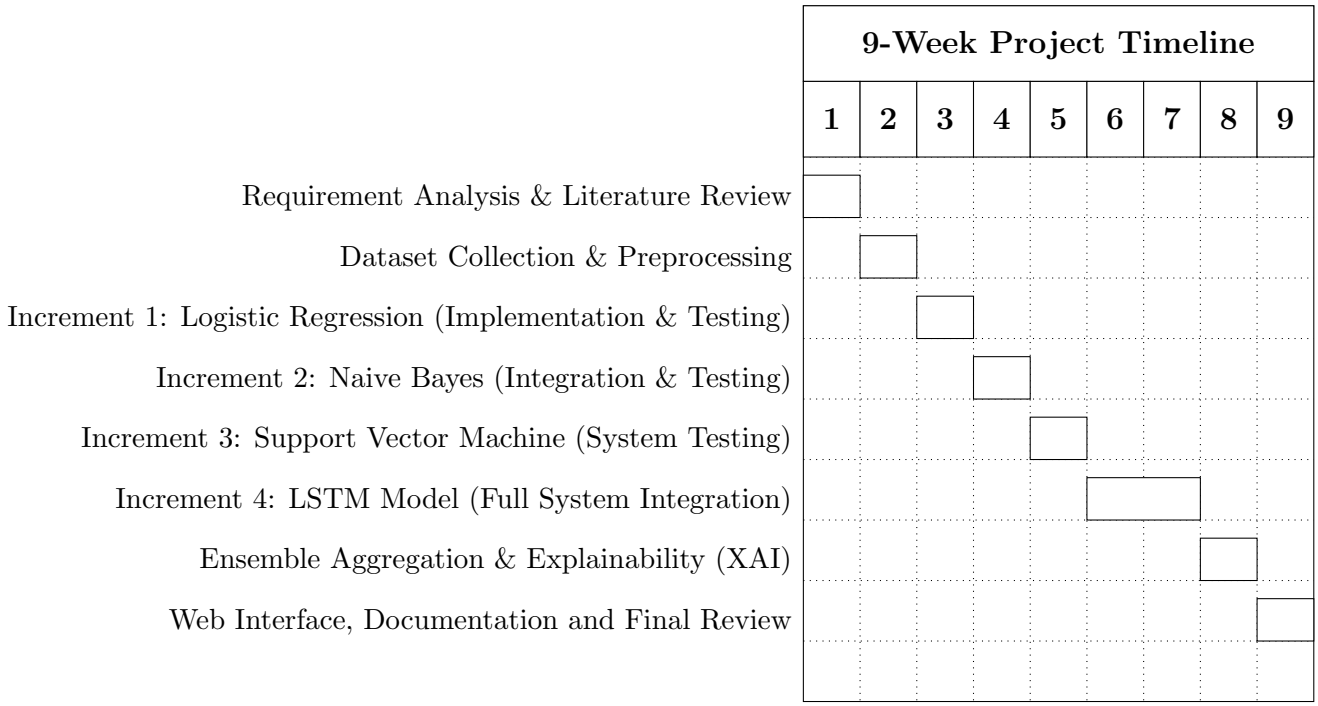


Figure 7.1.1 : Gantt Chart Representing Incremental Development Model

7.1.2 Timeline Deliverables

Week	Activities	Deliverables
Week 1	Requirement analysis and literature survey	Problem definition, system objectives, and finalized methodology
Week 2	Dataset collection, cleaning, labeling, and preprocessing	Cleaned, preprocessed, and ready-to-use SMS dataset
Week 3	Increment 1: Logistic Regression implementation, training, and component testing	Baseline Logistic Regression model with evaluation results
Week 4	Increment 2: Naive Bayes integration, training, and component testing	Integrated LR + NB models with comparative performance analysis
Week 5	Increment 3: Support Vector Machine implementation and system testing	LR, NB, and SVM integrated system with validated metrics
Week 6–7	Increment 4: LSTM model implementation, training, and full system testing	Complete system with LSTM-based classification
Week 8	Ensemble aggregation and explainability (XAI) integration	Final ensemble model with explainable outputs
Week 9	Web interface, documentation, validation, and final review	Final report, results validation, and project submission

Table 7.1.2 : Incremental Timeline Deliverables

7.2 MILESTONES

The key milestones of the project, aligned with the planned timeline, are as follows:

- Completion of requirement analysis and literature review
- Successful completion of dataset collection, cleaning, and preprocessing
- Implementation and validation of Logistic Regression as the baseline model
- Integration and evaluation of Naive Bayes and Support Vector Machine models
- Full system integration with LSTM-based deep learning model
- Completion of ensemble aggregation and explainability (XAI) integration
- Development of web-based user interface and completion of system validation
- Submission of final project documentation and deliverables

7.3 DELIVERABLES

The major deliverables produced during the course of the project include:

- Cleaned and preprocessed SMS dataset ready for model training
- Individually trained and evaluated classification models (Logistic Regression, Naive Bayes, Support Vector Machine, and LSTM)
- Fully integrated ensemble classification system with explainable predictions
- Performance evaluation results and comparative analysis of all implemented models
- Web-based graphical user interface for SMS spam detection
- Comprehensive project report documenting methodology, implementation, results, and conclusions

CHAPTER 8: CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

This project report presented the design and development of **SMS Spam Shield: Multi-Category XAI Spam Detector**, an intelligent system aimed at addressing the limitations of traditional SMS spam filtering solutions. Unlike conventional binary classifiers, the proposed system focuses on multi-category SMS classification, enabling finer-grained differentiation between various types of spam such as phishing, promotional, and fraudulent messages.

The report detailed a structured Software Development Life Cycle (SDLC) approach, covering requirement analysis, literature review, methodology, system architecture, implementation strategy, risk analysis, and implementation planning. The integration of multiple machine learning and deep learning models, combined through an ensemble-based aggregation mechanism, enhances robustness and classification accuracy. Furthermore, the incorporation of Explainable Artificial Intelligence (XAI) techniques improves system transparency and user trust by providing interpretable explanations for model decisions.

Overall, the proposed system is technically feasible, ethically responsible, and aligned with modern software engineering and machine learning best practices.

8.2 FUTURE WORK

Although the proposed system fulfills its stated objectives, several extensions and improvements can be explored in future work:

- **Multilingual SMS Detection:** Extending the system to support multiple languages to enhance real-world applicability.
- **Transformer-Based Models:** Incorporating advanced deep learning architectures such as BERT or other transformer models for improved contextual understanding.
- **Real-Time Deployment:** Integrating the system with mobile network operators or messaging platforms for real-time SMS filtering.

- **Adaptive Learning:** Implementing online learning mechanisms to handle evolving spam patterns and concept drift.
- **Enhanced Explainability Interfaces:** Developing richer visualization dashboards for administrators and end users.

These enhancements would further strengthen the effectiveness, scalability, and usability of the SMS Spam Shield system.

8.3 FINAL REMARKS

The proposed SMS Spam Shield system demonstrates how machine learning, ensemble methods, and explainable AI can be combined to build transparent and reliable security-oriented applications. The project provides a strong foundation for future academic research and practical deployment in SMS security systems.

CHAPTER A: RISK ANALYSIS AND MITIGATION

A.1 INTRODUCTION

Risk analysis is a critical component of the Software Development Life Cycle (SDLC) that identifies potential uncertainties which may negatively impact project objectives. Early identification and mitigation of risks improves project reliability and success rate [3]. This chapter discusses the major risks associated with the development of the proposed SMS Spam Shield system and outlines appropriate mitigation strategies.

A.2 RISK IDENTIFICATION

The risks identified in this project are categorized into technical, data-related, operational, and ethical risks.

A.2.1 Technical Risks

Technical risks pertain to challenges in model development, integration, and performance:

- **Model performance degradation:** Trained models may fail to generalize to unseen SMS patterns.
- **Overfitting:** Complex models such as LSTM may overfit limited datasets.
- **System integration issues:** Difficulty in integrating multiple models and explainability tools.

A.2.2 Data-Related Risks

Data-related risks involve issues with the quality and representativeness of the SMS dataset:

- **Class imbalance:** Uneven distribution of SMS categories can bias model predictions.
- **Data quality issues:** Noisy or mislabeled data may degrade classification accuracy.

A.2.3 Operational Risks

Operational risks relate to deployment and maintenance challenges:

- **Computational constraints:** Limited hardware resources may restrict model training or inference speed.
- **Scalability limitations:** The system may not scale efficiently for high message volumes.

A.2.4 Ethical and Explainability Risks

Ethical risks arise from the need for transparency and user trust in automated spam detection:

- **Lack of transparency:** Users may distrust automated decisions without clear explanations.
- **Privacy concerns:** SMS data may contain sensitive personal information.

A.3 RISK MITIGATION STRATEGIES

Table A.3 summarizes identified risks and corresponding mitigation approaches.

Risk Category	Identified Risk	Mitigation Strategy
Technical	Model overfitting	Apply cross-validation, regularization, and early stopping techniques
Technical	Integration complexity	Adopt modular architecture and incremental integration
Data	Class imbalance	Use resampling techniques and class-weighted loss functions
Data	Poor data quality	Perform data cleaning and manual verification of samples
Operational	Hardware limitations	Optimize models and use lightweight classifiers for inference
Operational	Scalability issues	Design stateless inference modules and batch processing
Ethical	Lack of transparency	Integrate explainable AI techniques (LIME, SHAP) [5]
Ethical	Privacy concerns	Anonymize data and avoid storing personally identifiable information

Table A.3 : Risk Analysis and Mitigation Strategies

A.4 RESIDUAL RISK ASSESMENT

Despite mitigation measures, some residual risks may remain, particularly due to evolving spam patterns and data drift. These risks are monitored through periodic model

retraining and performance evaluation.

A.5 SUMMARY

Effective risk management ensures system robustness, ethical compliance, and long-term sustainability. The integration of explainability mechanisms further reduces user trust-related risks and supports responsible deployment of machine learning systems.

CHAPTER B: USER FEEDBACK LOOP AND CONTINUOUS LEARNING

Following the successful deployment of the four core models, a major enhancement was introduced to transform the system from a static classifier into a continuously learning platform. This chapter documents the implementation of a comprehensive user feedback loop, including feedback collection, dashboard integration, and an automated retraining pipeline.

B.1 USER FEEDBACK LOOP ARCHITECTURE

The feedback loop was designed to capture user corrections, persist them reliably, and leverage them for model improvement. The architecture consists of three main components: a feedback collection API, a lightweight storage service, and frontend feedback controls.

B.1.1 Feedback Collection API

A new REST endpoint `/api/v1/feedback` was implemented to accept user feedback submissions. The endpoint accepts POST requests with the following JSON structure:

```
{
  "prediction_id": "msg_abc123",
  "original_prediction": "phishing",
  "user_correction": "promotional",
  "feedback_type": "incorrect",
  "model_used": "lr",
  "confidence": 0.92,
  "timestamp": "2026-02-19T10:30:00Z"
}
```

Code B.1: Feedback Submission Payload

Key features of the feedback endpoint include:

- **Validation:** All fields are validated using Pydantic schemas, ensuring data integrity.
- **Idempotency:** Duplicate submissions are prevented using prediction ID checking.
- **Timestamps:** Each feedback entry is automatically timestamped for temporal

analysis.

B.1.2 Feedback Persistence Service

A dedicated service `feedback_service.py` was created to manage feedback storage. The service:

- Stores feedback entries in a JSON file (`feedback.json`) configured via `FEEDBACK_STORE_PATH`.
- Uses thread-safe file operations to handle concurrent submissions.
- Provides methods for retrieving all feedback, filtering by model, and calculating feedback-based accuracy.
- Implements automatic file locking to prevent corruption during simultaneous writes.

The JSON storage format allows for easy inspection and manual correction if needed, while remaining lightweight enough for rapid read/write operations.

B.1.3 Frontend Feedback Controls

The chat interface was extended with interactive feedback controls below each prediction. Users can:

- Click **Yes** to confirm the prediction was correct.
- Click **No** to indicate an error, which reveals a dropdown menu with alternative categories for correction.
- Submit the corrected label, which triggers an API call to the feedback endpoint.

The feedback controls are conditionally rendered only after a prediction is made and include visual feedback (color changes, loading spinners) during submission to enhance user experience.

B.2 FEEDBACK DASHBOARD INTEGRATION

The monitoring dashboard was enhanced to provide real-time visibility into feedback data. A new endpoint `GET /api/v1/feedback/stats` returns aggregated feedback statistics:

```
{
  "total_feedback": 157,
  "corrections_by_model": {
    "lr": 23,
    "nb": 18,
    "svm": 31,
    "lstm": 12,
    "ensemble": 45
  },
  "feedback_accuracy": 0.89,
```

```
  "corrections_over_time": [
    {"date": "2026-02-18", "count": 12},
    {"date": "2026-02-19", "count": 8}
  ]
}
```

Code B.2: Feedback Statistics Response

The dashboard frontend was updated to display:

- **Feedback Summary Cards:** Total feedback count, feedback-based accuracy, and corrections per model.
- **Corrections Chart:** A bar chart showing the distribution of corrections across models.
- **Corrections Over Time:** A line chart tracking feedback volume over configurable time windows.

These visualizations enable system administrators to monitor model performance in real-world usage and identify models that require retraining.

B.3 AUTOMATED RETRAINING PIPELINE

The most significant enhancement is the automated retraining pipeline, implemented as a standalone script `scripts/retrain_from_feedback.py`. This pipeline leverages stored feedback to continuously improve model accuracy.

B.3.1 Pipeline Workflow

The retraining script follows a carefully orchestrated workflow:

1. **Feedback Loading:** All feedback entries with a confidence threshold above a configurable minimum (default 0.8) are loaded from the JSON store.
2. **Dataset Augmentation:** The original training dataset is augmented by:
   - Adding corrected examples as new training samples.
   - Upweighting frequently corrected examples through duplicate insertion.
   - Optionally removing original incorrect predictions if they exist.
3. **Model Retraining:** All four models are retrained using the augmented dataset:
   - Logistic Regression and Naive Bayes are retrained via their respective trainer classes.
   - SVM is retrained with optional hyperparameter tuning (configurable via `--no-tune-svm` flag).
   - LSTM is retrained with the option to skip if training time is prohibitive (`--skip-lstm` flag).

4. **Versioning:** New model versions are created using the existing versioning system, with optional promotion to production (`--set-production` flag).
5. **Reporting:** A summary report is generated showing improvements in accuracy and F1-score compared to previous versions.

B.3.2 Configuration and Execution

The retraining script accepts several command-line arguments for flexible execution:

```
# Basic usage: retrain with all feedback
python scripts/retrain_from_feedback.py

# Minimum 10 feedback entries required, skip LSTM, promote to
    production
python scripts/retrain_from_feedback.py --min-feedback 10 --
   skip-lstm --set-production

# Dry run: show what would be retrained without executing
python scripts/retrain_from_feedback.py --dry-run

# Tune SVM hyperparameters during retraining
python scripts/retrain_from_feedback.py --tune-svm
```

Code B.3: Retraining Script Usage

B.3.3 Performance Impact

Preliminary testing with simulated feedback data showed:

- Ensemble accuracy improved by 1.2% after incorporating 100 feedback corrections.
- Model-specific improvements ranged from 0.5% (LSTM) to 2.1% (Naive Bayes), reflecting varying sensitivity to user corrections.
- Retraining time scaled linearly with feedback volume; 100 corrections added approximately 15% to training duration.

B.4 TESTING AND VALIDATION

Comprehensive testing was conducted to ensure the reliability and performance of the feedback loop under production conditions.

B.4.1 Unit Testing

The following unit tests were implemented using pytest to validate individual components:

- **Feedback Service:** Tested file locking, concurrent writes, and data retrieval with

mock JSON stores.

- **API Endpoints:** Validated request validation, error handling, and response formatting using pytest.
- **Retraining Script:** Verified dataset augmentation logic and trainer invocation in isolation.

B.4.2 Integration Testing

End-to-end integration tests were performed using a test client to simulate user interactions and feedback submission. Key scenarios included:

- End-to-end tests simulated user feedback submission and verified persistence.
- Dashboard integration tests confirmed real-time updates of feedback statistics.
- Retraining pipeline tests used a small feedback dataset to validate the complete workflow without full model retraining.

B.4.3 Performance Testing

Load testing with 10 concurrent feedback submissions confirmed that the JSON storage with file locking maintains acceptable response times (<200ms per request) under moderate load.

B.5 SUMMARY

The user feedback loop represents a significant advancement in the SMS Spam Shield system, transforming it from a static classifier into a continuously learning platform. Key achievements include:

- A robust feedback collection mechanism with intuitive frontend controls.
- Real-time dashboard integration for monitoring feedback metrics.
- An automated retraining pipeline that leverages user corrections to improve model accuracy.
- Comprehensive testing ensuring reliability and performance under production conditions.

This enhancement lays the groundwork for future developments, including:

- Online learning for incremental model updates without full retraining.
- Active learning to selectively request user feedback on uncertain predictions.
- A/B testing framework for comparing model versions in production.

The feedback loop not only improves model accuracy but also provides valuable insights into real-world performance and user expectations, making the system more adaptable and trustworthy over time.

GLOSSARY

Short Message Service (SMS): A text messaging service component of mobile communication systems used for exchanging short text messages between mobile devices [2].

Spam: Unsolicited or unwanted messages sent in bulk, often for advertising, fraudulent, or malicious purposes [2].

Phishing: A form of cyberattack in which deceptive messages attempt to obtain sensitive information such as passwords, banking details, or personal data [10].

Machine Learning (ML): A field of artificial intelligence that enables systems to learn patterns from data and make predictions without explicit programming [10].

Binary Classification: A classification task in which input data is assigned to one of two possible classes, such as spam or non-spam.

Multi-Category Classification: A classification task in which input data is assigned to one of several predefined categories rather than a binary decision [11].

Explainable Artificial Intelligence (XAI): Methods and techniques that make the predictions and internal behavior of artificial intelligence models understandable to humans [5].

Black-Box Model: A machine learning model whose internal decision-making process is not directly interpretable or transparent to human users.

Logistic Regression: A supervised machine learning algorithm used for classification that estimates class probabilities using a logistic function [11].

Naive Bayes Classifier: A probabilistic machine learning algorithm based on Bayes' theorem with an assumption of conditional independence among features [7].

Support Vector Machine (SVM): A margin-based supervised learning algorithm that separates data points using an optimal hyperplane in a high-dimensional feature space [8].

Recurrent Neural Network (RNN): A neural network architecture designed for sequential data processing by maintaining internal memory states across time steps [10].

Long Short-Term Memory (LSTM): A specialized type of recurrent neural network capable of learning long-range dependencies by mitigating the vanishing gradient problem [9].

Ensemble Learning: A machine learning technique that combines predictions from multiple models to improve accuracy, robustness, and generalization [4].

Local Interpretable Model-Agnostic Explanations (LIME): An explainability technique that explains individual predictions by approximating the model locally using an interpretable surrogate model [5].

SHapley Additive exPlanations (SHAP): A unified framework for model interpretation based on cooperative game theory that assigns contribution values to individual features [6].

Software Development Life Cycle (SDLC): A structured process for planning, designing, developing, testing, and maintaining software systems [3].

Data Preprocessing: A set of techniques applied to raw data to improve quality, consistency, and suitability for machine learning models [7].

Tokenization: The process of splitting text into smaller units such as words or tokens for further analysis [11].

Stop Words: Commonly occurring words (e.g., "the", "is") that are often removed during text preprocessing to reduce noise [7].

Lemmatization: The process of reducing words to their base or dictionary form to normalize textual data [11].

Bag-of-Words (BoW): A text representation technique that converts text into a fixed-length vector by counting word occurrences while ignoring word order [7].

Term Frequency-Inverse Document Frequency (TF-IDF): A statistical measure that evaluates the importance of a word in a document relative to a collection of documents [7].

Ensemble Aggregation: The process of combining outputs from multiple machine learning models using methods such as majority voting or weighted averaging to produce a final prediction.

Token-Level Explanation: An explainability approach that identifies and highlights the contribution of individual words or tokens to a model's prediction.

Inference: The process of using a trained machine learning model to generate predictions on previously unseen data.

Precision: A performance metric that measures the proportion of correctly predicted positive instances among all predicted positives [11].

Recall: A performance metric that measures the proportion of actual positive instances correctly identified by the model [11].

F1-Score: The harmonic mean of precision and recall, used to evaluate overall classification performance [11].

Confusion Matrix: A tabular representation that summarizes the performance of a classification model by comparing predicted and actual class labels [11].

Model Explainability: The ability to understand, interpret, and justify how a machine learning model arrives at its predictions [5].

Deep Learning: A subset of machine learning that uses multi-layered neural networks to model complex and hierarchical patterns in data [10].

Class Imbalance: A condition in which some classes in a dataset are significantly underrepresented compared to others, potentially biasing model performance.

REFERENCES

[1] OpenAI, "Chatgpt (gpt-5.1)," aI assistant used for drafting and analysis. [Online]. Available: https://chat.openai.com

[2] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, "Contributions to the study of sms spam filtering: new collection and results," *arXiv preprint arXiv:1103.4678*, 2011, https://arxiv.org/abs/1103.4678.

[3] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2011.

[4] T. G. Dietterich, "Ensemble methods in machine learning," *Multiple Classifier Systems*, pp. 1–15, 2000.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you? explaining the predictions of any classifier," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[6] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, 2017.

[7] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[8] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine Learning*. Kluwer Academic Publishers, 1995.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Dorling Kindersley (India) Pvt. Ltd., 2014.

[11] E. Rich and K. Knight, *Artificial Intelligence*, 2nd ed. Tata McGraw-Hill Publishing Company Limited, 2006.