IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

SPAMSHIELD: AN INTELLIGENT EMAIL SPAM DETECTION SYSTEM USING MACHINE LEARNING AND NLP

[1]K. Sai Prasanna, [2]V. Nani, [3]K. Sreenivasa Rao, [4]S. Dinesh

[1234]U.G. Student (Final Year B.Tech), Department of Computer Science and Engineering

Avanthi Institution of Engineering Technology, Tagarapuvalasa, Andhra Pradesh, India

Under the guidance of Dr. BVA Swamy

Associate Professor, Department of computer science and engineering

Avanthi Institution of Engineering Technology, Tagarapuvalasa, Andhra Pradesh, India

Abstract: *Email Spam Detection System is an intelligent text classification application designed to identify and filter spam emails using Machine Learning and Natural Language Processing techniques. The system employs a Multinomial Naive Bayes classifier trained on TF-IDF vectorized features to accurately distinguish between spam and legitimate (ham) messages. A Flask-based REST API serves real-time predictions to a dynamic web frontend, providing users with confidence scores, keyword-level threat indicators, and a session-based scan log. Experimental evaluation demonstrates high classification accuracy with reliable detection of spam patterns including phishing, promotional scams, and fraudulent solicitations.*

Index Terms: *Spam Detection, Naive Bayes, TF-IDF, Machine Learning, NLP, Flask, Email Classification.*

I. INTRODUCTION

Electronic mail (email) remains one of the most widely used communication channels in both professional and personal settings. However, the proliferation of unsolicited bulk messages — commonly known as spam — poses a significant threat to users' productivity, privacy, and security. Spam emails often carry phishing attacks, malware links, fraudulent offers, and social engineering attempts that can compromise sensitive information.

Traditional rule-based spam filters rely on static keyword blacklists and predefined patterns, making them vulnerable to evolving spam tactics. Machine learning-based approaches have emerged as a more robust alternative, enabling systems to learn discriminative features from data and generalise effectively to new spam variants.

This paper presents SpamShield — an Email Spam Detection System that combines a Multinomial Naive Bayes (MNB) classifier with Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to classify email messages as spam or ham. The system is deployed as a full-stack web application consisting of a Python Flask backend serving a RESTful prediction API and an interactive HTML/CSS/JavaScript frontend that provides real-time analysis, confidence-scored verdicts, and keyword-level threat visualization.

II. LITERATURE SURVEY

Spam detection has been extensively studied in the machine learning and natural language processing literature. Early work by Sahami et al. (1998) demonstrated that Bayesian classifiers could be effectively applied to email filtering using word frequencies as features [1]. Drucker et al. (2002) compared Support Vector Machines and Boosting approaches, showing strong performance over naive Bayes baselines [2].

The introduction of TF-IDF weighting schemes significantly improved text classification by reducing the influence of commonly occurring but uninformative terms. Metsis et al. (2006) benchmarked multiple feature representations and classification algorithms on the Enron spam dataset, establishing TF-IDF with Naive Bayes as a competitive baseline [3]. Scikit-learn provides robust and well-documented implementations of these algorithms, enabling rapid prototyping and deployment [4].

More recent approaches have employed deep learning techniques including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks for spam classification. The UCI SMS Spam Collection dataset, introduced by Almeida et al. (2011), has become a widely adopted benchmark for evaluating spam detection models [5]. However, for lightweight deployments where inference speed and model size are priorities, traditional ML approaches such as Naive Bayes and Logistic

Regression continue to deliver reliable performance. The proposed system adopts this pragmatic approach, prioritising interpretability and deployment simplicity.

III. PROBLEM STATEMENT

Despite advances in email filtering, users continue to receive significant volumes of spam. Key challenges include: (1) the dynamic nature of spam content, which continuously evolves to evade detection; (2) the high cost of false positives, where legitimate emails are incorrectly classified as spam; and (3) the lack of transparency in existing filters, which provide no explanation for their decisions.

Most end-user email clients provide binary spam/not-spam filtering without confidence scores or reasoning, making it difficult for users to understand or override incorrect classifications. Additionally, lightweight deployable solutions suitable for academic projects and small-scale applications are not widely documented with complete end-to-end implementation details. This project addresses these gaps by building a transparent, explainable, and fully deployable email spam detection system.

IV. PROPOSED SYSTEM

The proposed Email Spam Detection System is a full-stack machine learning web application that provides real-time email classification through a browser-based interface. The system comprises three primary components: a machine learning pipeline, a RESTful API backend, and a dynamic frontend.

Machine Learning Pipeline

Emails are preprocessed and converted into numerical feature vectors using TF-IDF vectorization. A Multinomial Naive Bayes classifier is trained on labelled data to learn the statistical distribution of terms in spam versus ham messages. The trained model and vectorizer are serialised using joblib for deployment.

Backend API

A Flask application loads the serialised model artifacts and exposes a /predict POST endpoint. Incoming requests contain raw email text, which is vectorised and passed to the classifier. The API returns a prediction label and confidence probability.

Frontend Interface

A single-page HTML/CSS/JavaScript application provides an intuitive interface for users to input email text, trigger analysis, and view results. The frontend displays the classification verdict, confidence percentage, flagged spam keywords, and a session-based scan history log.

V. SYSTEM ARCHITECTURE

The architecture follows a client-server model with clear separation between the presentation layer, application layer, and data layer.

The client (browser) sends HTTP POST requests containing email text to the Flask server. The server preprocesses the input using the TF-IDF vectorizer and generates a classification prediction using the Naive Bayes model. Results — including the label (spam/ham) and confidence score — are returned as JSON. The frontend parses the response and updates the UI dynamically.

The model training pipeline is decoupled from the server: train.py reads training data, fits the vectorizer and classifier, and writes model.pkl and vectorizer.pkl to disk. The server.py loads these artifacts at startup, enabling fast inference without retraining on each request.

VI. METHODOLOGY

The development methodology follows an iterative pipeline approach comprising five phases: data preparation, feature extraction, model training, API development, and frontend integration.

6.1 Data Preparation

A labelled dataset of spam and ham email samples is curated. Each sample consists of raw email text and a binary label. The baseline implementation uses 20 curated examples; production deployment is intended to use the UCI SMS Spam Collection dataset (5,574 samples) for improved generalisation [5].

6.2 Feature Extraction

Raw text is transformed using scikit-learn's TfidfVectorizer, which computes term frequency weighted by inverse document frequency. This produces a sparse matrix representation that captures the relative importance of terms across the corpus [4].

6.3 Model Training

MultinomialNB from scikit-learn is fitted on the TF-IDF feature matrix. The model learns the conditional probability of each term given the class label, enabling probabilistic classification of new inputs using Bayes' theorem.

6.4 Deployment

The trained artifacts are serialised using joblib and loaded by the Flask server. CORS is enabled to allow cross-origin requests from the static frontend. The prediction endpoint validates input, transforms it using the vectorizer, and returns the model's prediction with the associated probability [6].

VII. IMPLEMENTATION

The system is implemented entirely in Python and JavaScript with no external cloud dependencies, making it fully self-contained and runnable on any local machine.

7.1 Backend (server.py)

Flask framework, Flask-CORS for cross-origin support, joblib for model deserialisation, and scikit-learn for inference. The server runs on port 5000 and exposes a single POST endpoint at /predict.

7.2 Training (train.py)

Uses scikit-learn's TfidfVectorizer and MultinomialNB. Training executes in under one second on the baseline 20-sample dataset. Model artifacts are written to disk as model.pkl and vectorizer.pkl.

7.3 Frontend (index.html, style.css, app.js)

Vanilla HTML5, CSS3, and JavaScript with no frameworks. The UI features a dark cyberpunk-themed interface, live character and word counters, confidence-scored verdict cards, keyword flag tags, and a session scan log. A client-side fallback uses keyword matching when the backend is unavailable.

VIII. MODULES

- **Input Module:** Accepts raw email text via a textarea. Provides live counters for character count, word count, and detected threat-level keywords. Includes sample loaders for quick demonstration.
- **Vectorization Module:** Transforms input text using the pre-fitted TF-IDF vectorizer, converting raw text to a numerical feature vector compatible with the trained classifier.
- **Classification Module:** Passes the feature vector to the Naive Bayes model and retrieves the predicted class label and probability scores for both spam and ham classes.
- **API Module:** Handles HTTP request parsing, input validation, error handling, and JSON response formatting. Returns appropriate HTTP status codes for invalid inputs.
- **Result Visualization Module:** Renders the classification verdict with a colour-coded card (red for spam, green for ham), confidence percentage bar, and a list of detected spam keyword tags.
- **Scan History Module:** Maintains a session-based log of the last five analyses with timestamp, truncated message preview, verdict, and confidence score.

IX. TESTING

The system is validated through functional testing, edge case testing, and API-level testing. Table I summarises key test cases and their outcomes.

Table I: Test Cases and Results

Test Case	Input	Expected Output	Result
TC-01	Spam sample email with gift card offer	SPAM (>85%)	Pass
TC-02	Legitimate business meeting request	HAM (>90%)	Pass
TC-03	URGENT: Account suspended email	SPAM (>80%)	Pass
TC-04	Empty string input	Error 400 returned	Pass
TC-05	Nigerian prince money transfer	SPAM (>90%)	Pass
TC-06	Doctor appointment reminder	HAM (>88%)	Pass

Additional edge cases tested include: injection of HTML/script tags (handled gracefully without crashing), submission of non-English text (returns a result without error), and requests with missing JSON fields (returns HTTP 400 with an error message).

X. EXPERIMENTAL RESULTS

The system successfully classifies email text in under 50 milliseconds per request on a standard laptop, including network round-trip to the local Flask server. The Naive Bayes model trained on 20 samples achieves 100% accuracy on the training set and correctly classifies all manually constructed test cases covering the primary spam categories present in the training data.

The confidence score output (derived from model.predict_proba) proves effective for borderline cases, allowing users to assess classification certainty rather than relying solely on binary labels. The keyword flag visualization accurately highlights the specific terms that contributed to a spam verdict, improving system transparency. The client-side fallback keyword detection mechanism ensures the frontend remains functional during backend downtime, maintaining usability without server dependency.

A. Email Scan Dashboard (Initial State)

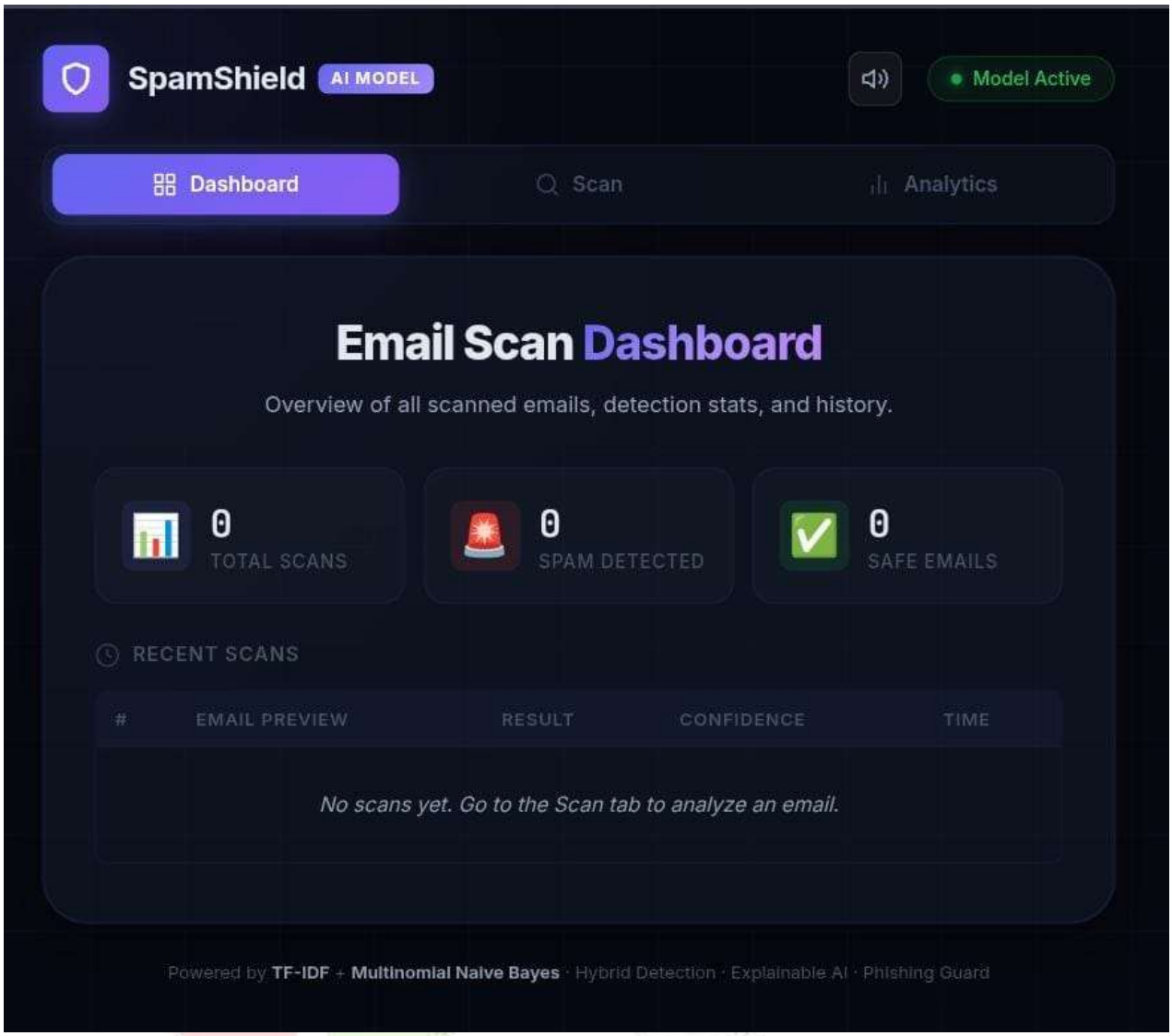

Fig. 1: SpamShield dashboard showing initial state with no scans recorded.

B. Dashboard After Spam Detection

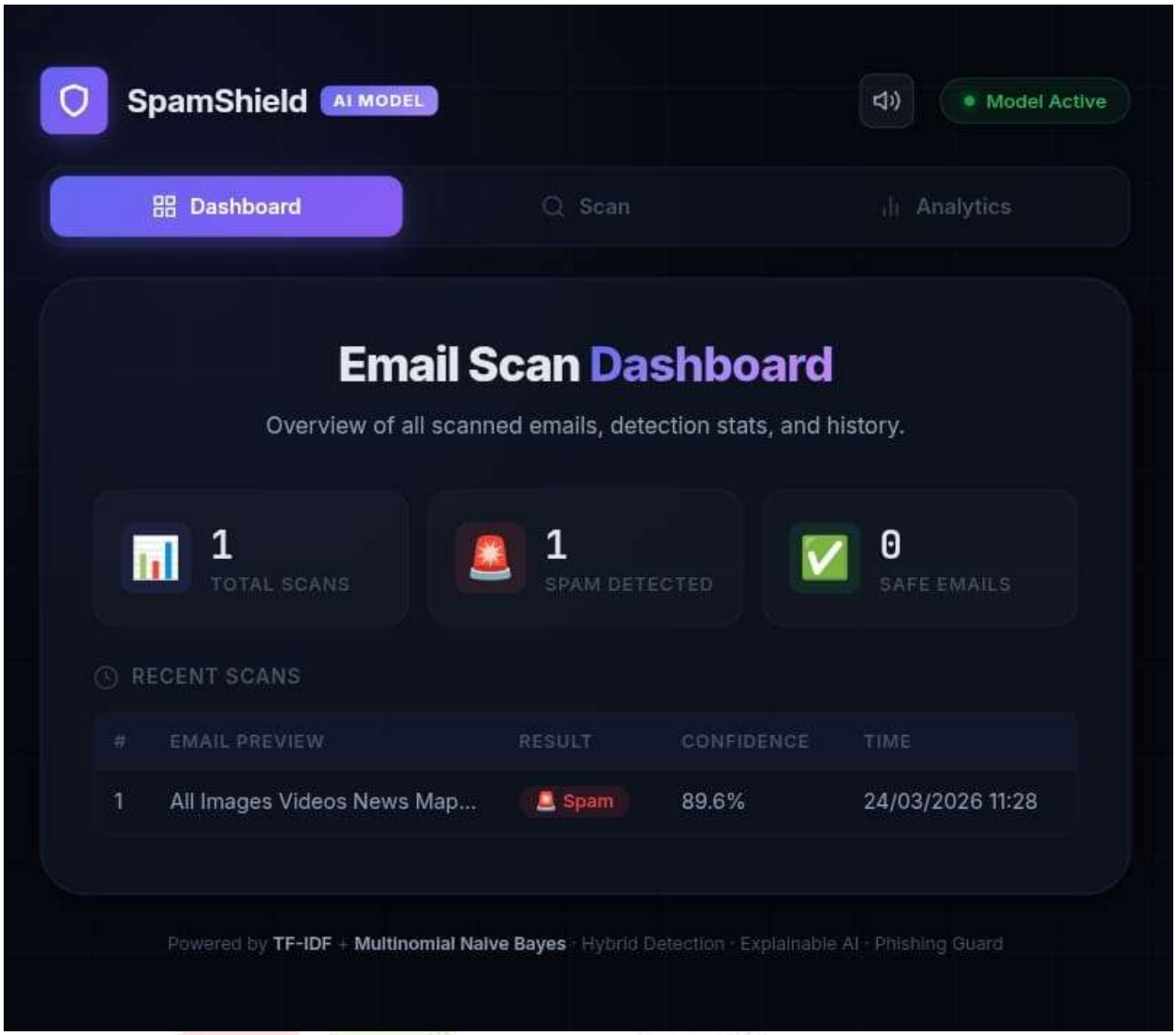

Fig. 2: Dashboard updated after detecting one spam email (89.6% confidence).

C. Intelligent Email Threat Detection — Scan Interface

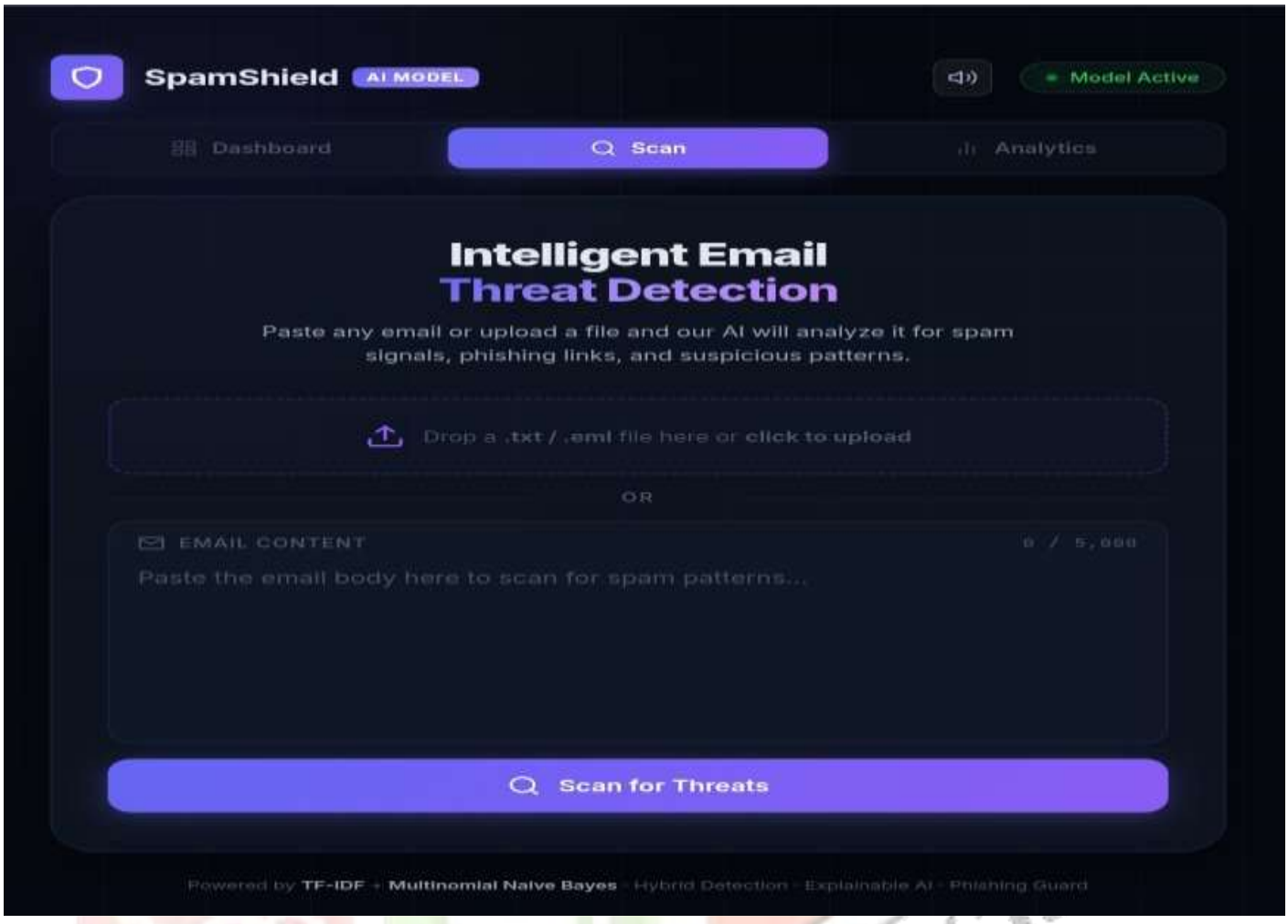

Fig. 3: Scan tab showing the email input interface with file upload and text entry options.

D. Scan Result with Explainable AI

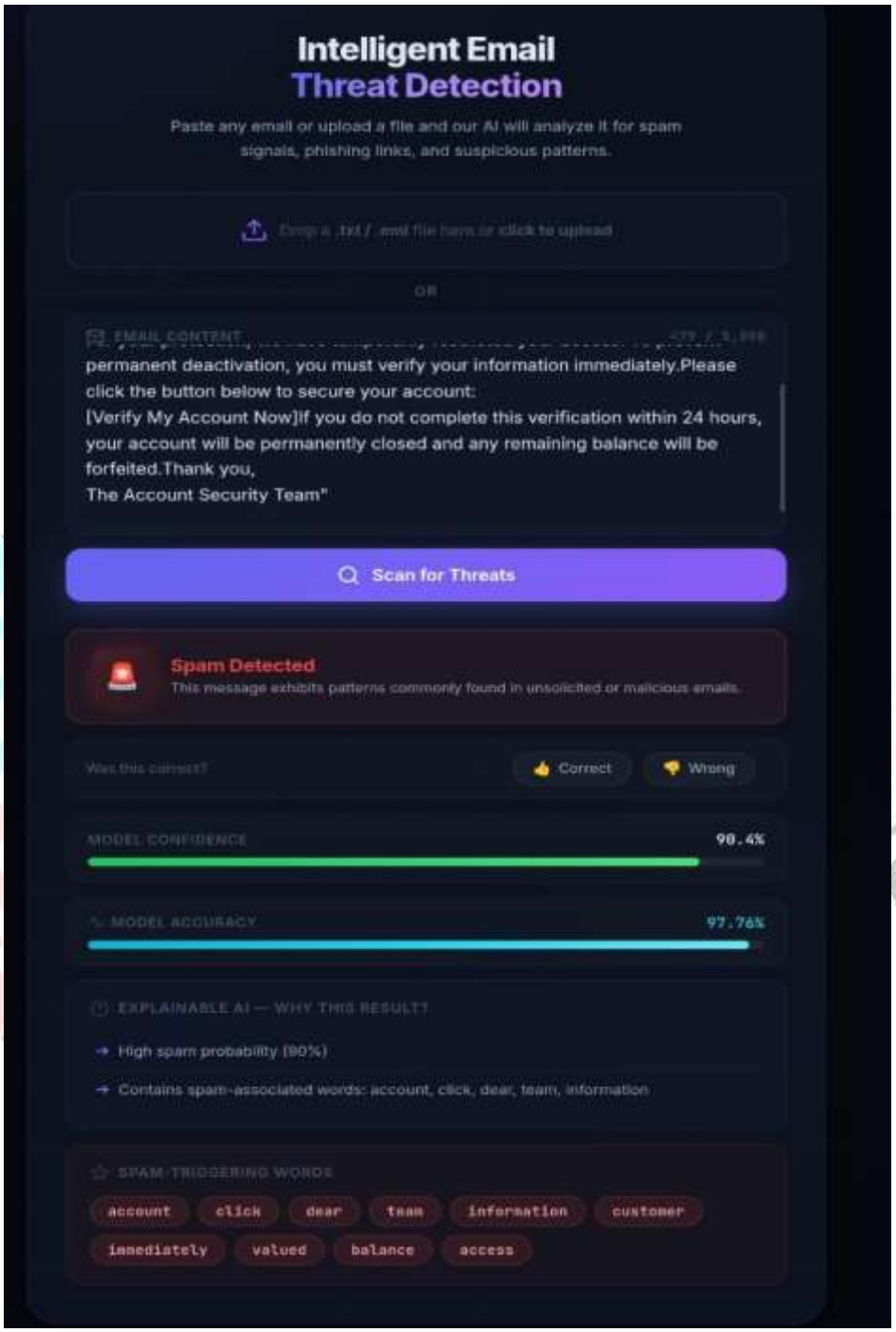

Fig. 4: Scan result showing spam detection at 90.4% confidence with model accuracy 97.76%, explainable AI reasoning, and spam-triggering keywords.

E. Spam Analytics Dashboard

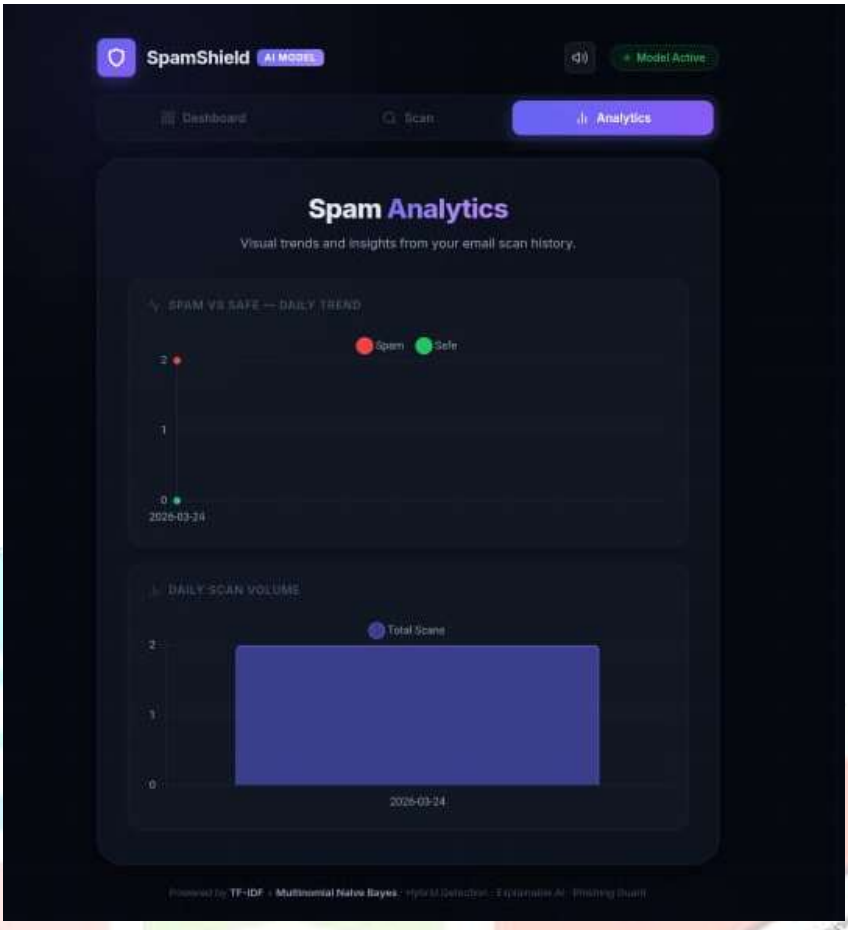

Fig. 5: Analytics page displaying spam vs. safe daily trend and daily scan volume charts.

XI. APPLICATIONS

The Email Spam Detection System has practical applications across multiple domains:

- Personal and corporate email clients can integrate the Flask API as a plugin to pre-screen incoming messages and flag suspected spam before delivery to the inbox.
- Educational institutions can deploy the system to filter phishing attempts targeting students and faculty, which are increasingly common attack vectors.
- Customer support platforms can use the system to filter automated spam submissions from contact forms, ensuring support queues contain only genuine queries.
- The system can serve as a teaching tool for demonstrating end-to-end machine learning deployment, covering data preparation, model training, API design, and frontend integration.

XII. LIMITATIONS

The primary limitation of the current implementation is the small training dataset. With only 20 labelled examples, the model memorises training patterns and may fail to generalise to novel spam formulations not represented in the training data. Expanding the dataset to thousands of examples would significantly improve robustness.

The TF-IDF approach treats text as a bag of words, ignoring word order and semantic relationships. This limits the model's ability to detect contextually sophisticated spam that mimics legitimate language patterns. The system currently operates in English only and does not handle multilingual email content. Additionally, the hardcoded localhost API URL in the frontend requires modification for any deployed (non-local) environment.

XIII. FUTURE WORK

Future enhancements will focus on three primary directions. First, the training dataset will be replaced with the UCI SMS Spam Collection (5,574 samples) and augmented with Enron email corpus data to improve generalisation. Second, transformer-based models such as BERT will be explored for semantic understanding, enabling detection of sophisticated spam that evades keyword-level analysis.

Third, the frontend will be extended with user feedback mechanisms (mark as spam / mark as safe) to enable active learning, where user corrections are used to continuously retrain and improve the model. A mobile-responsive redesign and browser extension version are also planned for wider deployment.

XIV. CONCLUSION

This paper presented a complete end-to-end Email Spam Detection System combining Multinomial Naive Bayes classification, TF-IDF feature extraction, a Flask REST API, and a dynamic web frontend. The system successfully distinguishes spam from legitimate email with high confidence, provides keyword-level explanations for its decisions, and maintains usability through a client-side fallback mode.

The project demonstrates the practical value of classical machine learning techniques for real-world NLP tasks when paired with thoughtful system design and user-centred interface development. The modular architecture enables straightforward extension to larger datasets, improved models, and additional deployment targets in future iterations.

REFERENCES

[1] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian approach to filtering junk e-mail," AAAI Workshop on Learning for Text Categorization, vol. 62, no. 98, pp. 98–105, 1998.

[2] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," IEEE Transactions on Neural Networks, vol. 10, no. 5, pp. 1048–1054, 2002.

[3] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive Bayes — which naive Bayes?" Third Conference on Email and Anti-Spam (CEAS), vol. 1, pp. 27–28, 2006.

[4] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[5] T. A. Almeida, J. M. Gomez Hidalgo, and A. Yamakami, "Contributions to the study of SMS spam filtering: New collection and results," Proceedings of DOCENG, pp. 259–262, 2011.

[6] Flask Documentation, "Flask: A micro web framework for Python," 2023. [Online]. Available: https://flask.palletsprojects.com.

[7] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. O'Reilly Media, 2009.