1. INTRODUCTION

In today's digital-first world, customer support plays a vital role in maintaining user satisfaction and loyalty. With businesses receiving thousands of support emails daily, the need for intelligent systems to streamline and automate email triage has become more crucial than ever. Manual classification is time-consuming, error-prone, and inefficient—especially when the emails contain sensitive information like names, emails, card numbers, and contact details.

This project addresses this challenge by building a robust, end-to-end email classification system designed specifically for customer support teams. The system classifies incoming emails into predefined categories such as *Incident*, *Request*, or *Billing Issues* using traditional NLP and machine learning techniques. It also ensures data privacy by performing automatic **PII masking**—replacing sensitive fields with placeholders before any processing or inference.

The final system is lightweight, interpretable, and deployable via a public API using **Gradio** and **Hugging Face Spaces**, making it ideal for real-world integration. It is built entirely with non-LLM techniques to showcase the power of classical NLP approaches, while still maintaining high utility, accuracy, and compliance with privacy standards.

2. APPROACH

The solution was developed through a structured pipeline consisting of three core components: **PII masking**, **email classification**, and **API deployment**. The focus was to ensure privacy compliance, prediction accuracy, and production-readiness using lightweight, explainable ML methods rather than black-box LLMs.

I. PII Masking

The first step in processing incoming emails is identifying and masking Personally Identifiable Information (PII) and PCI data. This is critical to prevent leakage of sensitive data during model training and inference.

We adopted a hybrid method:

- **Regex-based extraction** for well-defined fields such as:
    - Emails
    - Phone numbers
    - Credit/debit card numbers
    - Aadhar numbers
    - CVV and expiry dates
- **SpaCy NER (Named Entity Recognition)** for more context-sensitive entities like:
    - Full names
    - Dates of birth

Each identified entity is replaced with a label token (e.g., [full_name], [email]) and logged with its position and original value for audit and demasking purposes.

II. Text Vectorization and Classification

Masked emails are passed to a classification pipeline for categorization. We chose a **TF-IDF vectorizer** for transforming text into numerical features and a **Random Forest Classifier** for robust, interpretable predictions.

Key decisions:

- **TF-IDF max features limited to 1000** to reduce overfitting and model size.
- **Random Forest parameters** were tuned:
    - n_estimators = 50
    - max_depth = 10
- Final model accuracy was validated with cross-validation (optional).

The trained model is serialized using pickle into model.pkl and kept under 100MB to be GitHub-friendly.

III. System Deployment and Integration

We opted for **Gradio** as the UI framework for rapid prototyping and easy deployment on Hugging Face Spaces. The app.py file handles user input, invokes the masking and classification pipeline, and displays the output in the format specified by Akaike Technologies.

To ensure compatibility with Hugging Face:

- The SpaCy model en_core_web_sm is auto-downloaded at runtime.
- All dependencies are listed in requirements.txt.

3. MODEL TRAINING

The dataset ('emails.csv') includes labeled support emails. We trained a TF-IDF + RandomForestClassifier pipeline. To reduce model size, we limited TF-IDF features and reduced tree depth. The final model was saved as 'model.pkl' and loaded during API inference.

4. CHALLENGES & SOLUTIONS

Building a complete system for email classification with strict output formatting and PII protection posed several technical and practical challenges. Here's a breakdown of the major hurdles and how they were tackled:

I. Handling Varied PII Formats

Challenge: Emails contained PII in multiple unpredictable formats—dates like 12/03/1999, phone numbers with or without +91, card numbers with spaces or dashes, etc.

Solution:

- Designed robust **regular expressions** that captured all standard and edge-case formats.
- Used **SpaCy NER** for detecting entities like names and DOBs that aren't easily captured with regex.

II. Strict API Response Format

Challenge: Akaike required a specific JSON format with exact keys, nesting, and PII entity positions.

Solution:

- Carefully tracked the start and end positions of each masked entity using re.finditer instead of re.sub.
- Manually constructed the response dictionary to match the format instead of relying on auto-serialization.

III. Deployment Environment Limitations

Challenge: Hugging Face Spaces restrict runtime and memory. Also, en_core_web_sm (SpaCy) isn’t pre-installed.

Solution:

- Added os.system("python -m spacy download en_core_web_sm") to app.py so the model is installed at runtime.
- Optimized code and imports for faster load time and smaller memory usage.

III. GitHub File Size Limit (100MB)

Challenge: The trained model (model.pkl) was initially too large to commit to GitHub.

Solution:

- Reduced **TF-IDF max features** from default to 1000.
- Used a **simplified RandomForestClassifier** with fewer estimators and lower depth.
- Achieved a final model under 20MB without major accuracy loss.

5. CONCLUSION

The project is fully functional and deployed on Hugging Face Spaces.

API accepts a POST request with an email body, masks PII, classifies the email, and returns a structured

JSON response.

Project includes training script, model, API, and deployment setup.

6. DEPLOYMENT LINK

https://pavanstunner-emails.hf.space/?logs=container&__theme=system&deep_link=7CdwPeVqmW4