VIDYASHILP UNIVERSITY

ARTIFICIAL INTELLIGENCE

TEAM: T5

FRAUDSHIELD AI: DETECTING DIGITAL ARREST SCAM CALLS IN 60 SECONDS

Submitted on : 31/03/2025

Submitted by: **Group Members : Madiha Khan, Sabahath Taj Z, Trisha, Rakshitha, Nikhil, Chandan**

Table of Contents

1. Introduction

1.1. Project Overview

This report details the architecture, implementation, and operational capabilities of the "Secure Audio Analysis Portal" – a sophisticated, web-based application engineered to provide users with critical intelligence regarding the nature of recorded audio communications, primarily focusing on identifying potentially fraudulent or malicious phone calls. Developed using Python and leveraging the interactive capabilities of the Streamlit framework for an accessible and responsive user interface, the portal serves as a dedicated analytical tool for end-users.

At its technological core, the portal integrates Google's advanced gemini-1.5-flash large language model (LLM), harnessing its multimodal capabilities for nuanced audio interpretation. The system is designed to ingest user-uploaded audio files (handling common formats like WAV, MP3, M4A), perform necessary pre-processing including duration normalization (analyzing the crucial initial 60 seconds), and subsequently invoke the Gemini API for two primary functions:

- **Heuristic Threat Classification:** Employing a meticulously crafted, rule-based prompt, the AI analyzes the audio content for indicators characteristic of known fraudulent tactics (e.g., impersonation of authority, artificial urgency, sensitive data solicitation) and spam behaviors, classifying the call into distinct categories: 'Fraud', 'Spam', 'Normal', or 'Unclear/Empty'. This classification is accompanied by an AI-generated justification, providing context for the assessment. The prompt design explicitly accounts for potential variations in language, including English and common Indian languages, ensuring broader applicability.
- **Automated Transcription:** The portal provides high-fidelity audio transcription, generating both an English transcript and attempting to capture the transcription in the originally detected language, offering users a textual representation of the call content.

Supporting this core AI functionality is a robust backend infrastructure featuring secure user authentication via bcrypt hashing, persistent storage of analysis results within an SQLite database (linking classifications and transcriptions to user accounts and specific files), and an integrated email notification system using SMTP for optionally alerting designated recipients (e.g., a security team or fraud monitoring address) upon detection of 'Fraud' or 'Spam' incidents, as well as facilitating user feedback submission.

1.2. Problem Statement: The Escalating Challenge of Phone-Based Fraud and Spam

The impetus for this project stems directly from the escalating prevalence and alarming sophistication of telephonic fraud and unsolicited commercial communication (spam) that plague the modern digital landscape. Malicious actors increasingly exploit voice channels, employing advanced social engineering tactics, voice modulation, and AI-generated scripts to execute deceptive schemes. These range from:

- **Impersonation Scams:** Fraudsters posing as officials from law enforcement (e.g., "digital arrest" threats citing fake cybercrime violations), tax authorities, financial institutions (requesting OTPs or account details under false pretenses), or technical support representatives.
- **High-Pressure Sales & Extortion:** Aggressive tactics demanding immediate payment for fictitious debts, fines, or services, often coupled with threats of legal action, service disconnection, or account compromise.

- **Information Phishing:** Calls designed solely to extract sensitive personal identifiable information (PII) such as Aadhaar numbers, PAN details, passwords, or banking credentials for subsequent identity theft or financial fraud.

Simultaneously, the sheer volume of unsolicited spam calls (ranging from aggressive telemarketing to nuisance robocalls) degrades the user experience, erodes trust in voice communication, and consumes valuable time.

The critical significance of addressing this problem lies in the substantial risks faced by individuals:

- **Severe Financial Loss:** Victims can lose significant sums of money through direct fraudulent transfers or compromised accounts.
- **Identity Theft:** Stolen PII can lead to long-term financial and legal complications.
- **Emotional Distress:** The anxiety, fear, and violation associated with being targeted or victimized by scams can have profound psychological impacts.
- **Erosion of Trust:** The prevalence of malicious calls undermines legitimate uses of telecommunication.

Distinguishing expertly crafted fraudulent calls from legitimate interactions or even benign spam requires vigilance and often specific knowledge that the average user may lack, particularly when under pressure. This creates a critical vulnerability gap that necessitates accessible, rapid, and intelligent assessment tools. The "Secure Audio Analysis Portal" is conceived as a direct response to this challenge, aiming to empower users by providing actionable, AI-driven insights into the nature of their recorded calls, thereby mitigating risk and fostering a safer communication environment.

2. Problem Statement

2.1. Definition of the Problem: Exploitation of Voice Channels for Malicious and Intrusive Communication

The fundamental problem addressed by this project is the systemic exploitation of telephony and voice communication channels for perpetrating financial fraud and disseminating unsolicited, often intrusive, commercial content (spam). This issue transcends mere nuisance; it represents a significant vector for criminal activity and a pervasive source of disruption for individuals globally.

The challenge is characterized by several key dimensions:

- **Escalating Sophistication of Fraudulent Tactics:** Malicious actors are no longer reliant on rudimentary scams. Modern phone-based fraud increasingly involves sophisticated social engineering, psychological manipulation, and potentially AI-assisted techniques. This includes:
- **Convincing Impersonation:** Fraudsters skillfully mimic representatives of legitimate entities, including government agencies (e.g., Police, Cyber Crime divisions, Tax Departments, TRAI/DoT), financial institutions (banks, credit card security), major corporations (e.g., tech support, e-commerce logistics), and judicial bodies. The application's AI prompt design directly reflects the need to identify these specific impersonation scenarios (e.g., "Officer Sharma from Cyber Crime," "digital arrest," "bank account compromised").
- **Exploitation of Urgency and Fear:** Calls often create artificial high-pressure situations, leveraging threats of immediate legal action, arrest warrants, account freezing, service disconnection, or catastrophic data loss to impair rational judgment and compel compliance.
- **Targeted Information Phishing:** Beyond direct payment demands, calls are meticulously designed to illicit sensitive Personal Identifiable Information (PII) such as Aadhaar or PAN numbers, full bank account details, login credentials, and One-Time Passwords (OTPs), which are then used for identity theft or unauthorized financial access. The application specifically targets the detection of suspicious OTP requests as a critical fraud indicator.
- **Circumvention of Traditional Defenses:** Scammers utilize techniques like Caller ID spoofing and disposable numbers to bypass basic filtering mechanisms.
- **Difficulty in Real-Time Discernment:** For the average individual, distinguishing a sophisticated fraudulent call from a legitimate, albeit perhaps unexpected, communication in real-time is exceptionally difficult. The pressure tactics employed, combined with the plausibility of the impersonation, often leave little room for verification before potential damage occurs.
- **Pervasiveness of Spam:** Alongside targeted fraud, users are inundated with a high volume of unsolicited calls focused on aggressive telemarketing, unwanted surveys, or deceptive lead generation. While distinct from outright fraud, this spam contributes significantly to communication fatigue, erodes trust, and can occasionally serve as a gateway to more elaborate scams. The application's design acknowledges this by including 'Spam' as a distinct classification category.

2.2. Significance of the Problem: Protecting Financial, Psychological, and Infrastructural Integrity

Addressing this multifaceted problem is of paramount importance due to its severe and wide-ranging consequences:

- **Substantial Financial Harm:** Successful phone scams result in direct and often significant financial losses for victims, ranging from depleted savings accounts to unauthorized credit usage and long-term debt. The collective economic impact globally runs into billions of dollars annually.
- **Profound Psychological Impact:** Beyond financial loss, victimization leads to considerable emotional distress, including anxiety, fear, feelings of violation and embarrassment, and a lasting erosion of personal security and trust in others. Vulnerable populations, such as the elderly or less digitally literate individuals, are often disproportionately affected.
- **Erosion of Trust in Communication Infrastructure:** The constant barrage of fraudulent and spam calls undermines the reliability and trustworthiness of essential telecommunication services. This negatively impacts legitimate businesses, emergency services, and personal communications, as users become increasingly wary of answering calls from unknown numbers.
- **Resource Drain:** Individuals and organizations expend significant time and resources dealing with unwanted calls, attempting to verify suspicious communications, and recovering from successful attacks.

Therefore, the development of effective, accessible tools for identifying and classifying potentially harmful calls is not merely a convenience but a critical necessity for safeguarding individual well-being, protecting financial assets, and maintaining the integrity of vital communication networks in the face of increasingly sophisticated threats. This project directly confronts this need by offering an AI-driven analytical capability to empower users in this defense.

3. Objectives

The primary objectives established for the development and deployment of the "FraudShield AI" are as follows:

- **Develop a Secure, User-Centric Web Portal:** To create an accessible, intuitive, and secure web application using the Streamlit framework, providing a centralized platform for users to manage and analyze audio recordings.

- **Implement Robust User Authentication and Authorization:** To ensure secure access control through a mandatory registration and login system, employing industry-standard password hashing techniques (bcrypt) and persistent user session management.
- **Integrate Advanced AI for Audio Threat Classification:** To leverage the capabilities of Google's gemini-1.5-flash LLM via its API for the primary purpose of analyzing uploaded audio content (normalized to the initial 60 seconds) and classifying it according to a predefined, nuanced schema ('Fraud', 'Spam', 'Normal', 'Unclear/Empty') based on a sophisticated, custom-engineered prompt designed to recognize specific fraudulent indicators and differentiate them from legitimate communication patterns, with awareness of multilingual contexts (English and various Indian languages).
- **Provide High-Fidelity Automated Transcription:** To utilize the same Gemini AI integration to offer users accurate automated transcription of the processed audio segment, aiming to provide both an English translation and a transcript in the detected source language where feasible.
- **Establish Efficient Audio Pre-processing:** To implement a reliable audio processing pipeline using the Pydub library, capable of handling various common audio input formats, standardizing them (to WAV), and enforcing a maximum duration limit (60 seconds) for consistent and efficient AI analysis.
- **Ensure Persistent Storage of Analysis Artifacts:** To systematically record the outcomes of each analysis (including user identification, original filename, processed audio data, AI classification, AI justification, and timestamp) within a structured SQLite database for historical reference, potential review, and traceability.
- **Enable Proactive Alerting and Feedback Mechanisms:** To incorporate an email notification system (via SMTP) allowing users to optionally trigger alerts to a designated recipient upon classifying a call as 'Fraud' or 'Spam', and to provide a dedicated channel for users to submit operational feedback directly via email.
- **Deliver Actionable Insights and Clear User Feedback:** To present the results of AI analysis and transcription clearly within the UI, employing visual cues (e.g., color-coding for classification severity) and providing the AI's rationale to aid user understanding and decision-making.

4. Solution Overview

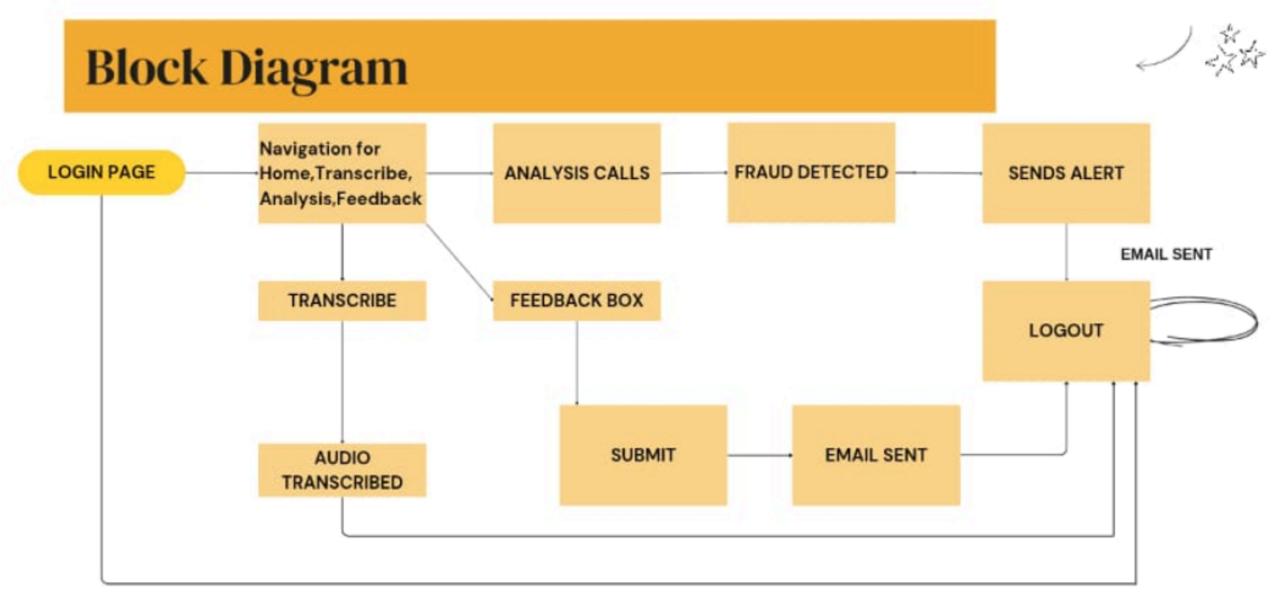
4.1. High-Level Description

The "FraudShield AI" is architected as an integrated Python-based web application, utilizing the Streamlit framework to deliver a dynamic and interactive user interface accessible via a web browser. The core value proposition lies in its ability to provide users with AI-driven analysis and transcription of uploaded audio recordings, specifically targeting the identification of fraudulent or spam phone calls.

Upon secure authentication (registration/login managed via bcrypt hashing and an SQLite user database), users can upload audio files. The backend system initiates a standardized workflow: the uploaded audio is first processed using the Pydub library to ensure compatibility, normalize format (to WAV), and truncate the data to the critical initial 60 seconds. This processed audio segment is then securely transmitted, via the google-generativeai library, to the Google Gemini API (gemini-1.5-flash model).

Depending on the user's chosen action ('Analyze' or 'Transcribe'), a specific, purpose-built prompt (FRAUD_ANALYSIS_PROMPT or TRANSCRIPTION_PROMPT) is sent alongside the audio data reference. The Gemini model performs the requested operation – either classifying the call based on the detailed heuristic rules embedded in the analysis prompt or generating a multilingual transcription. The textual response from the AI is received by the backend, parsed to extract relevant information (classification, justification, transcript), and then presented back to the user within the Streamlit interface using informative visual elements (e.g., status messages, color-coded results, interactive audio players, text areas). Concurrently, key details of the analysis, including the classification and reason, are persistently logged in the calls table of the SQLite database, linked to the user's account. For classifications deemed 'Fraud' or 'Spam', the user is provided an option to trigger an immediate email notification to a pre-configured administrative or monitoring address using Python's smtplib. A separate feedback mechanism also utilizes this email infrastructure.

4.2: Block Diagram



5. Technical Stack Used

The development and operation of the "FraudSheild AI" rely on a specific set of software technologies, programming libraries, external services, and system dependencies. The technical stack is outlined below:

5.1. Core Programming Language & Framework:

Programming Language: Python (Version 3.7+ recommended due to dependencies and modern syntax usage). Python serves as the foundation for the entire application logic, including backend processing, API interactions, and database management.

Web Framework: Streamlit (streamlit). This Python-based framework is utilized exclusively for building the interactive graphical user interface (GUI), managing application state, handling user inputs, and rendering dynamic content within a web browser environment.

5.2. Key Python Libraries:

AI Integration:

google-generativeai: The official Google client library for interacting with the Gemini family of large language models. Used for configuring the API connection, uploading audio data, making generative content requests (for both analysis and transcription), and managing API responses.

Audio Processing:

pydub: A crucial library for manipulating audio files. Employed here to load audio data from various formats (requiring FFmpeg backend), crop audio segments to a predefined maximum duration (MAX_AUDIO_DURATION_MS), and export audio into a standardized format (WAV) suitable for the AI model.

io: (Standard Library) Used extensively with pydub and file uploads to handle audio data as in-memory binary streams (BytesIO), avoiding the need for temporary disk files.

Database Management:

sqlite3: (Standard Library) Python's built-in module for interacting with SQLite databases. Used to initialize the database schema (users and calls tables), store user credentials, and log audio analysis results persistently in the calls.db file.

Authentication & Security:

bcrypt: A library dedicated to securely hashing passwords using the bcrypt algorithm. Utilized for storing user passwords in a non-reversible format and verifying credentials during login.

Email Notifications:

smtplib: (Standard Library) Provides the client-side implementation of the Simple Mail Transfer Protocol (SMTP). Used to establish secure connections (SSL/TLS via SMTP_SSL) to an email server (Gmail in this configuration) for sending notification and feedback emails.

email.mime: (Standard Library) Part of Python's email handling package, specifically email.mime.text.MIMEText and email.mime.multipart.MIMEMultipart, used for constructing properly formatted email messages (setting headers like From, To, Subject, and structuring the message body).

Utilities:

os: (Standard Library) Used for basic operating system interactions, potentially checking for file existence (though primarily base64 handles file reading here).

time: (Standard Library) Employed for introducing short delays (time.sleep) during UI feedback (e.g., after login success) and potentially for waiting loops (like checking Gemini file status). Also used for timestamping feedback emails.

base64: (Standard Library) Used specifically within the set_styles function to encode the background image file (background.jpg) into a base64 string, allowing it to be embedded directly within CSS for the background styling.

Pillow: (Likely Implicit Dependency) Although not directly imported in the main script, Pillow (Python Imaging Library fork) is often a dependency for Streamlit or libraries handling media, and might be implicitly required for full functionality or certain Streamlit features. Listed in requirements.txt for clarity.

5.3. Databases and APIs:

Database: SQLite. A lightweight, file-based relational database management system. The application uses a single database file (calls.db) to store user information and call analysis records.

External APIs: Google Gemini API. This is the core external service providing the advanced AI capabilities. The application specifically utilizes the gemini-1.5-flash model endpoint for its audio analysis and transcription tasks.

5.4. Runtime Environment & Dependencies:

Cloud Services: The Google Cloud Platform implicitly hosts the Gemini API infrastructure that the application interacts with. User audio data is temporarily uploaded to Google's servers during the analysis process via genai.upload_file.

External Software Dependency: FFmpeg. This is a critical external multimedia framework that must be installed on the host system where the Streamlit application is run. Pydub relies heavily on FFmpeg for decoding and encoding a wide range of audio formats beyond basic WAV. Without FFmpeg, the application will likely fail to process most uploaded audio file types (e.g., MP3, M4A, OGG, FLAC).

Hardware: Standard computer hardware capable of running Python 3, Streamlit, and FFmpeg. Requires sufficient RAM for handling audio processing in memory. Internet connectivity is essential for interacting with the Google Gemini API and the SMTP server.

6. System Architecture

6.1.Architecture Overview

FraudShield AI consists of multiple components that interact to process and classify call recordings efficiently. The system follows a structured workflow from user authentication to fraud detection and alert generation.

6.2. Explanation of Component Interactions

The system architecture is designed around a central backend logic controller within the app.py script, orchestrated by the Streamlit framework, and interacting with specialized modules and external services.

User Interaction & Frontend (User -> Streamlit UI -> Backend Controller):

- The user accesses the application via a web browser, interacting with the Streamlit UI.
- Streamlit handles rendering the web pages (Login, Register, Welcome, Analyze, Transcribe, Feedback), managing interactive widgets (buttons, file uploaders, text inputs), and maintaining the user's session state (st.session_state).
- User actions (e.g., button clicks, form submissions, file uploads) generate events that are captured by Streamlit and directed to the corresponding functions within the Backend Logic / Controller (app.py).

Authentication Flow (Controller -> AuthMod -> DB -> Controller):

- For Login/Registration, the Controller invokes the Authentication Module (authenticate, save_user).
- This module uses the bcrypt library to hash passwords (for registration) or verify submitted passwords against stored hashes (for login).
- It interacts directly with the users table in the SQLite Database to retrieve or store user credentials.
- The success or failure status is returned to the Controller, which updates the UI and session state accordingly.

Audio Processing Flow (Controller -> AudioProcMod -> FFmpeg -> AudioProcMod -> Controller):

- When a user uploads an audio file on the Analyze or Transcribe pages, the Controller receives the raw audio bytes.

- It passes these bytes to the Audio Processing Module (process_audio).
- This module utilizes the pydub library, which in turn invokes the external FFmpeg installation to decode the input audio format (if not WAV).
- pydub then performs operations like cropping the audio to the configured maximum duration (60 seconds) and exporting the result as WAV-formatted bytes in memory.
- The processed audio bytes (and its MIME type) are returned to the Controller.

AI Analysis/Transcription Flow (Controller -> AIMod -> Ext_Gemini -> AIMod -> Controller):

- The Controller takes the processed audio bytes and selects the appropriate prompt (FRAUD_ANALYSIS_PROMPT or TRANSCRIPTION_PROMPT).
- It passes this data to the AI Interaction Module (get_gemini_response).
- This module uses the google-generativeai library to:
- Upload the audio bytes to Google Cloud (managed by the library).
- Send an API request to the Google Gemini API (gemini-1.5-flash), including the prompt and the reference to the uploaded audio.
- Receive the text-based response from the Gemini API.
- Attempt to delete the temporarily uploaded file from Google Cloud.
- If analysis was requested, the raw response is further processed by parse_fraud_analysis_response within the AI Module to extract the structured classification and reason.
- The final result (parsed analysis or raw transcription) is returned to the Controller.

Data Persistence Flow (Controller -> DBMod -> DB):

- After successful AI analysis, the Controller gathers the relevant details (user email, filename, processed audio bytes, classification, reason).
- It invokes the Data Persistence Module (save_audio_data).
- This module uses the sqlite3 library to execute an SQL INSERT statement, storing the analysis record in the calls table of the SQLite Database.

Notification & Feedback Flow (Controller -> EmailMod -> Ext_SMTP):

- If the user triggers a 'Send Alert' (for Fraud/Spam) or 'Submit Feedback', the Controller invokes the Email Notification Module (send_fraud_report, send_feedback_email, which uses send_email).

- This module constructs the appropriate email message using email.mime.
- It then uses smtplib to establish a secure connection (SSL) to the configured External SMTP Server (e.g., Gmail) and transmits the email using the provided credentials (SENDER_EMAIL, EMAIL_APP_PASSWORD).

Configuration Access (Controller -> Config):

- Throughout its operation, the Controller and various modules access configuration parameters (API keys, email addresses, database file path) defined near the top of the app.py script (Config). Ideally, these sensitive values would be externalized (e.g., using st.secrets).

UI Update (Controller -> UI):

- After backend operations complete (e.g., analysis finished, email sent, authentication successful), the Controller updates st.session_state and/or uses Streamlit functions (st.success, st.error, st.write, st.rerun) to refresh the Streamlit UI, displaying results, status messages, or navigating the user to a different page.

7. Implementation Details

7.1. Steps Followed to Build the Solution (Inferred Process)

The construction of the "FraudShield AI" likely followed an iterative development process, building upon core functionalities incrementally:

Foundation & Environment Setup:

- Initialized a Python project environment.
- Installed core dependencies: streamlit, google-generativeai.
- Set up the basic Streamlit application structure (app.py, main function, if _name_ == "_main_":).
- Implemented basic page configuration (st.set_page_config) and initial placeholder UI elements.

Core AI Integration (Proof of Concept):

- Configured the Google Gemini AI client (genai.configure, selecting gemini-1.5-flash).
- Implemented a basic file uploader (st.file_uploader).

- Created the initial get_gemini_response function to handle API calls, likely starting with simple text prompts before integrating audio.
- Integrated audio handling, possibly starting with only WAV files, then adding pydub for broader format support (process_audio). This stage likely involved debugging the audio upload process (genai.upload_file) and handling file states ('PROCESSING', 'ACTIVE', 'FAILED').
- Developed initial versions of the FRAUD_ANALYSIS_PROMPT and TRANSCRIPTION_PROMPT.
- Integrated audio playback (st.audio) and display of raw AI responses.

Refining Analysis & Transcription:

- Iteratively refined the FRAUD_ANALYSIS_PROMPT based on test results to improve classification accuracy and adherence to the desired schema (Fraud/Spam/Normal/Unclear). Added detailed rules, examples, and multi-language considerations.
- Implemented parse_fraud_analysis_response to reliably extract structured data from the AI's potentially variable output, including fallback logic.
- Refined the TRANSCRIPTION_PROMPT to request both English and original language output and handle silent/unclear audio gracefully.
- Implemented audio cropping (pydub) in process_audio to enforce the 60-second limit, ensuring consistent processing time and potentially managing API costs/limits. Added error handling for audio decoding issues (CouldntDecodeError).

User Authentication & Database Integration:

- Set up the SQLite database (init_db) with users and calls tables.
- Integrated the bcrypt library for secure password hashing.
- Implemented user registration (save_user) and login (authenticate) logic.
- Built the Login/Register UI using st.tabs and st.form.
- Integrated session management (st.session_state) to track login status (logged_in, user_email) and control access to core application features.
- Implemented the data saving logic (save_audio_data) to store analysis results in the calls table.

Notification & Feedback System:

- Implemented the core email sending function (send_email) using smtplib and email.mime, configuring it for Gmail SMTP (SSL). This likely involved obtaining an App Password.
- Created wrapper functions (send_fraud_report, send_feedback_email) for specific notification types.
- Added UI elements (buttons for 'Send Alert', feedback form) to trigger these email functions.

UI/UX Enhancements & State Management:

- Implemented the multi-page navigation using the sidebar and st.session_state['current_page'].
- Added visual feedback mechanisms like st.spinner during long operations and status messages (st.success, st.error, st.warning).
- Applied custom styling using CSS injected via st.markdown, including the background image (get_base64_of_bin_file, set_styles) and specific button colors for better visual cues.
- Refined state management, ensuring results were linked to specific uploads (using dynamic keys like f'analysis_result_{file_name}') and that state was appropriately cleared on logout.

Configuration & Finalization:

- Consolidated configuration variables (API keys, email settings, DB path) at the beginning of the script (though noting the security issue of hardcoding).
- Added comments and performed general code cleanup.
- Created requirements.txt (inferred) and documented the FFmpeg dependency.

7.2.Innovations or Unique Approaches Used

- Sophisticated Prompt Engineering: The FRAUD_ANALYSIS_PROMPT represents a significant effort in prompt engineering. Its detailed, rule-based structure, inclusion of specific scam typologies (digital arrest, bank impersonation), explicit handling of multi-language audio input, and strict output formatting instructions aim to guide the LLM towards highly specific and reliable classification behavior.
- Integrated Multimodal AI Workflow: The application seamlessly combines audio processing (pydub), multimodal AI interaction (gemini-1.5-flash handling audio input), NLP classification based on audio content, and speech-to-text transcription within a single, unified user experience delivered via Streamlit.
- Targeted Model Selection: Utilizing gemini-1.5-flash suggests a deliberate choice aiming for a balance between the advanced reasoning capabilities needed for the nuanced classification task and the lower latency/cost benefits compared to potentially more powerful (but slower/pricier) models like Gemini Pro or Ultra.

- Stateful Result Handling in UI: Using dynamically generated keys in st.session_state (e.g., f'result_key_{file_name}') allows the UI to correctly display results corresponding to specific file uploads, even if multiple files are processed sequentially within the same session before a full page refresh.
- Proactive Resource Management (Gemini): The inclusion of genai.delete_file after processing demonstrates good practice in managing resources on the cloud platform where the audio file is temporarily stored for analysis by Gemini.
- In-Memory Audio Handling: The consistent use of io.BytesIO for passing audio data between the uploader, pydub, and potentially the AI client library avoids writing temporary files to the disk, making the process potentially cleaner and faster.

7.3:Challenges faced and they were overcome

- **Challenge: Backend Integration – Connecting Frontend & Backend**

Solution: Faced difficulty linking the frontend (Streamlit UI) with the backend (SQLite, AI model). Resolved by properly structuring function calls, ensuring that user authentication and audio processing interact seamlessly through API-like functions within Streamlit.

- **Challenge: Email Alerts Not Being Sent**

Solution: Verified SMTP server configurations, ensuring correct host, port, and authentication details. Also, tested with different email providers to bypass security restrictions and enabled “Less Secure Apps” access for SMTP-based services.

8. LLM and AI Integration

8.1:LLM Used

The core artificial intelligence capabilities of the "FraudShield AI" are powered by Google's Gemini family of models. Specifically, the application is configured to utilize the gemini-1.5-flash model.

This is explicitly defined in the code during the initialization phase:

```
# --- Initialize Gemini ---

try:

    genai.configure(api_key=GEMINI_API_KEY)

    # Explicit selection of the gemini-1.5-flash model
    gemini_model = genai.GenerativeModel('gemini-1.5-flash')

    print("Gemini configured successfully.")

except Exception as e:

    # Error handling if configuration fails

    st.error(f"Fatal Error: Could not configure Google Gemini AI...")

    gemini_model = None
```

Use code with caution.

Python

The choice of gemini-1.5-flash suggests a strategic decision prioritizing faster response times and potentially lower operational costs compared to larger Gemini variants (like Pro), while still leveraging a powerful multimodal model capable of handling the required audio analysis and transcription tasks effectively.

8.2. How AI Components Were Integrated

The integration of AI functionalities, primarily Speech-to-Text (Transcription) and Natural Language Processing (NLP for Classification), is achieved through a well-defined workflow orchestrated by the Python backend (Myfraud.py) interacting with the Google Gemini API via the google-generativeai Python library.

Configuration and Client Initialization:

The process begins by configuring the google-generativeai library with the necessary API key (genai.configure(api_key=GEMINI_API_KEY)).

An instance of the target model (gemini-1.5-flash) is created using genai.GenerativeModel(...), providing an object (gemini_model) to interact with.

Multimodal Input Handling (Audio Upload):

Since Gemini is a multimodal model capable of processing inputs beyond text, the application handles audio uploads specifically. Before making an analysis or transcription request, the processed audio bytes (WAV format, max 60s) are uploaded to Google's infrastructure using genai.upload_file(path=audio_file_object, mime_type=processed_mime_type).

This upload process returns a file object representing the audio on Google's servers. The application includes logic to wait (while uploaded_file.state.name == "PROCESSING": ...) until this file is ready ('ACTIVE') before proceeding.

Task Invocation via Prompts and API Calls:

A central function, get_gemini_response(prompt, audio_bytes, mime_type), encapsulates the interaction with the Gemini API.

Crucially, the specific AI task (Transcription or Classification) is determined by the prompt argument passed to this function.

1. **Speech-to-Text (Transcription):** When the user requests transcription (transcribe_page), the get_gemini_response function is called with the TRANSCRIPTION_PROMPT. This prompt explicitly instructs the gemini-1.5-flash model to transcribe the speech content of the referenced audio file, requesting output in both English and the detected original language.
2. **NLP for Classification:** When the user requests analysis (fraud_analysis_page), the function is called with the detailed FRAUD_ANALYSIS_PROMPT. This prompt leverages the NLP capabilities of the model. It instructs Gemini to interpret the meaning, intent, context, and specific keywords/phrases within the audio, apply the complex set of provided rules (related to impersonation, urgency, demands, etc.), and classify the call accordingly ('Fraud', 'Spam', 'Normal', 'Unclear/Empty'). This goes beyond simple transcription; it requires understanding and reasoning based on the audio content.

The core API call is made using gemini_model.generate_content([prompt, uploaded_file]), passing both the instructional text prompt and the reference to the previously uploaded audio file.

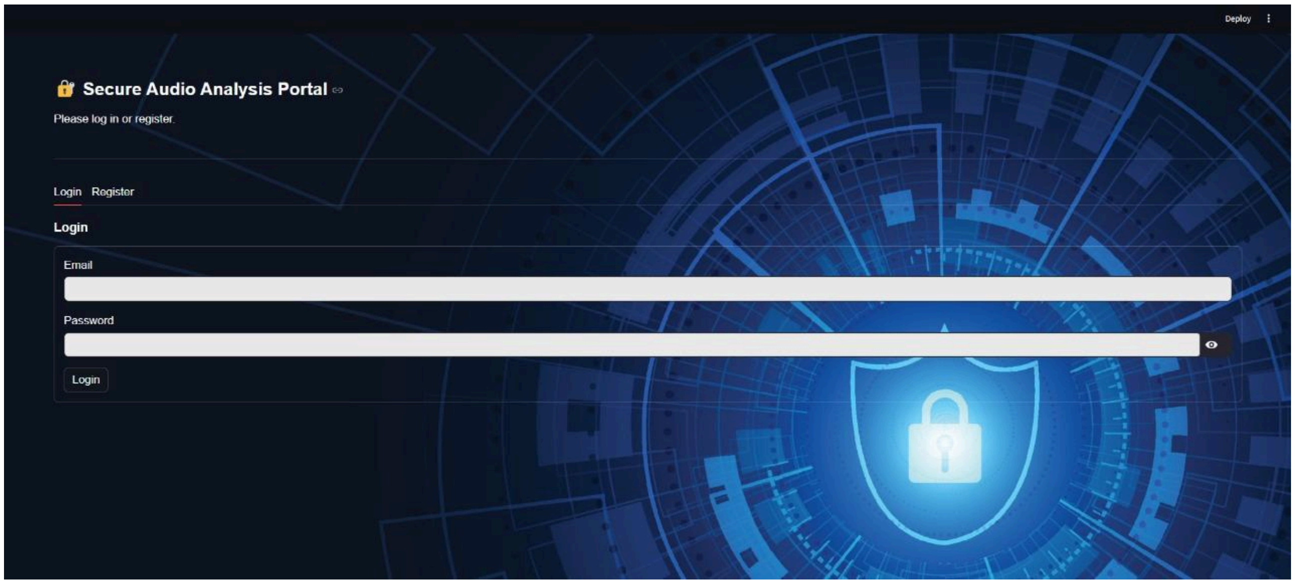
Response Processing and Resource Cleanup:

1. The get_gemini_response function retrieves the textual response generated by the Gemini model (response.text.strip()).
2. For classification tasks, this raw text response is further processed by the parse_fraud_analysis_response function, which applies string manipulation and basic NLP logic to extract the structured classification label and the justification text, handling potential variations in the AI's output format.
3. The get_gemini_response function also includes logic to attempt deletion of the uploaded audio file from Google's servers (genai.delete_file) after the API call completes, managing temporary resources.
4. In essence, the integration relies on using the google-generativeai SDK to handle the complexities of API communication and multimodal data transfer, while leveraging carefully crafted text prompts to direct the powerful, general-purpose gemini-1.5-flash

model to perform specific Speech-to-Text and NLP-driven classification tasks on the provided audio input.

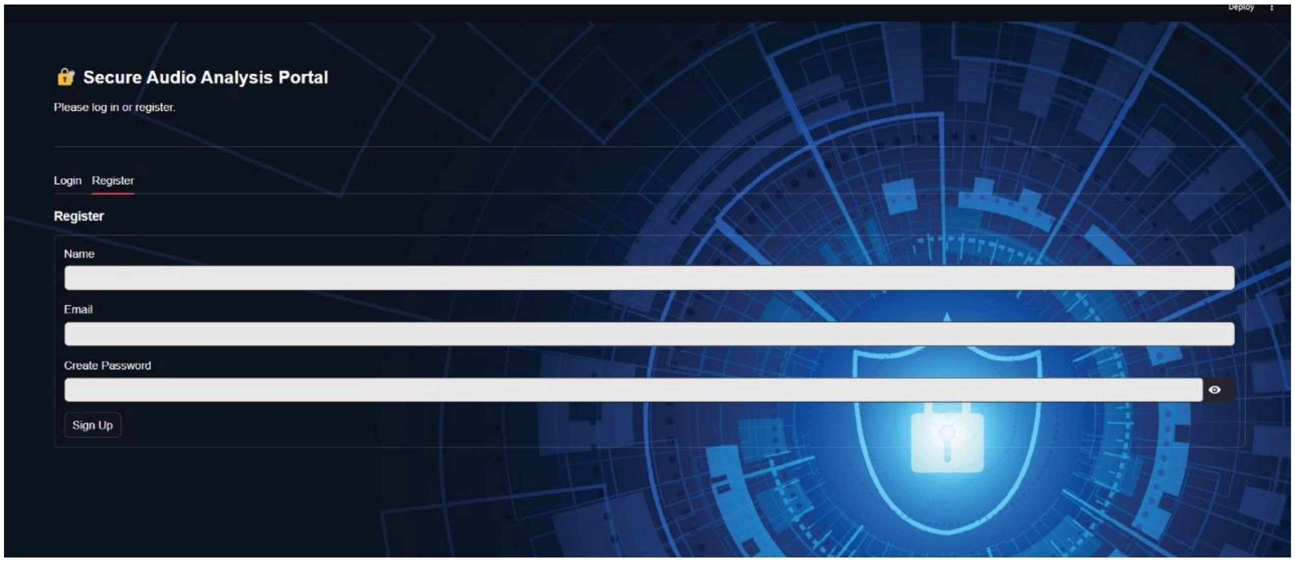
9. Frontend & UI Design

9.1.Snapshots of User Interface(UI)



Secure Audio Analysis Portal

Please log in or register.



Secure Audio Analysis Portal

Please log in or register.

9.2.Explanation of UX decisions

The User Experience (UX) Design of this Secure Audio Analysis Portal login page follows a clean and professional layout with a focus on security. Here's a breakdown of the design elements:

1. Visual Design & Theme

- The dark-themed background with a futuristic circuit pattern and a glowing lock icon reinforces the idea of security and encryption.
- The blue color scheme aligns with cybersecurity aesthetics, making users feel confident about data protection.

2. Simplicity & Clarity

- The login page is minimalistic with only essential options: Login and Register, reducing distractions.
- The use of clear labels (Email, Password, Login button) ensures easy navigation for users.

3. Input Fields & Usability

- The email and password fields are large and well-spaced, improving readability and usability.
- The password field has an eye icon, allowing users to toggle password visibility for better user control.

4. Accessibility & Navigation

- The Login and Register tabs are placed at the top, providing an intuitive navigation experience.
- The "Deploy" option at the top right suggests additional functionality, possibly for administrators or developers.

5. User Trust & Security Cues

- The lock icon in the background emphasizes security, reassuring users about data protection.
- The padlock emoji in the title enhances the branding and theme of secure audio analysis.

Overall, the UX design of this login page is simple, professional, and security-focused, making it effective for a secure audio analysis platform.

10. Code Structure & Execution Guide

10.1. Code Structure Explanation

The current implementation of the "Secure Audio Analysis Portal" adopts a monolithic script structure. All Python code, encompassing the Streamlit user interface definition, backend logic for authentication, audio processing, AI interactions (Google Gemini), database operations (SQLite), email notifications (SMTP), configuration settings, and utility functions, is contained within a single file named Myfraud.py.

Consequently, the essential file organization for running this application is minimal and flat:

```
project-directory/

|

├── Myfraud.py           # The single, comprehensive Python script contains all application logic.

├── background.jpg        # Background image file referenced by Myfraud.py for UI styling.

├── calls.db             # SQLite database file (auto-generated by Myfraud.py on first run if
absent).

└── requirements.txt      # Required: Lists external Python dependencies needed by Myfraud.py.
```

Use code with caution.

project-directory/: A directory chosen by the user to hold the project files.

Myfraud.py: The core and sole Python source file. It must reside in this directory.

background.jpg: The image file required for the UI background. It must be placed in the same directory as Myfraud.py so the script can locate and load it.

calls.db: The SQLite database. The script is configured to create and access this file within the same directory it is executed from.

requirements.txt: Although the code is in one file, it relies on external libraries. This file is crucial for listing these dependencies (Streamlit, google-generativeai, pydub, bcrypt, Pillow) and must reside in the same directory to facilitate easy installation via pip install -r requirements.txt.

Implications of Single-File Structure:

- **Simplicity for Small Projects:** This structure can be straightforward for initial development or smaller-scale applications as all code is readily accessible in one place.
- **Reduced Modularity:** It inherently lacks modularity. Separating concerns (UI, database, AI logic, utilities) into different files or modules typically enhances maintainability, testability, and collaboration on larger projects.
- **Maintainability Challenges:** As the application grows in complexity, navigating, modifying, and debugging a single large file can become increasingly challenging.
- **Namespace Management:** All functions and global variables reside in the same namespace, increasing the potential for naming conflicts if not carefully managed.

10.2. Steps to Set Up and Run the Code

Follow these steps precisely to set up the development environment and execute the application:

1. Prerequisites Installation:

1. **Python:** Ensure you have Python installed (version 3.7 or newer is recommended). You can download it from python.org. Verify installation by opening a terminal or command prompt and typing python --version or python3 --version.
2. **pip:** Python's package installer, usually included with Python installations. Verify with pip --version or pip3 --version.
3. **FFmpeg:** This is a critical external dependency required by the pydub library for processing most audio formats (MP3, M4A, OGG, FLAC, etc.).
4. Download FFmpeg from the official site: https://ffmpeg.org/download.html
5. Follow the installation instructions for your operating system (Windows, macOS, Linux).

Crucially, ensure the ffmpeg executable is added to your system's PATH environment variable so that pydub can find and execute it. You can test this by typing ffmpeg -version in your terminal – it should output version information without errors.

2. Obtain Project Files:

1. Create the root project directory (e.g., mkdir secure-audio-analysis-portal).
2. Navigate into the directory (cd secure-audio-analysis-portal).
3. Save the provided Python code as app.py inside this directory.
4. Place the background.jpg image file in the same directory.

3. Create requirements.txt:

Inside the project directory, create a file named requirements.txt and add the following lines:

- streamlit
- google-generativeai
- pydub
- bcrypt
- Pillow

Use code with caution.

Txt

4. Install Python Dependencies:

Open your terminal or command prompt, ensure you are in the project's root directory (secure-audio-analysis-portal/).

Run the following command to install the libraries listed in requirements.txt:

```
pip install -r requirements.txt
```

Use code with caution.

(Use pip3 if pip is linked to an older Python version on your system).

5.Configure Sensitive Credentials (CRITICAL STEP):

You have two options here. Option A is insecure but reflects the current code. Option B is the strongly recommended secure method.

Option A: Hardcoding (Insecure - As per original code):

1. Open the app.py file in a text editor.
2. Locate the configuration section near the top.
3. Replace the placeholder string for GEMINI_API_KEY with your actual Google Gemini API Key.
4. Replace the placeholder strings for SENDER_EMAIL, RECEIVER_EMAIL, and EMAIL_APP_PASSWORD with your actual Gmail address, the recipient address for alerts, and a 16-digit Gmail App Password. (Generate an App Password in your Google Account

settings under Security -> 2-Step Verification -> App passwords. Do NOT use your regular Google password.)

WARNING: Storing credentials directly in the code is a major security risk, especially if the code is shared or version controlled.

Option B: Using Streamlit Secrets (Recommended & Secure):

1. Create the .streamlit directory inside your project folder: mkdir .streamlit
2. Inside .streamlit, create a file named secrets.toml.
3. Add your credentials to secrets.toml in the following format:

```
# .streamlit/secrets.toml

GEMINI_API_KEY = "YOUR_ACTUAL_GEMINI_API_KEY"

SENDER_EMAIL = "your_sender_email@gmail.com"

RECEIVER_EMAIL = "your_recipient_email@example.com"

EMAIL_APP_PASSWORD = "your_16_digit_gmail_app_password"
```

Use code with caution.

Toml

Now, modify app.py. Replace the direct assignments:

```
# Replace this:

# GEMINI_API_KEY = "AIza..."

# SENDER_EMAIL = "madihakhan83100@gmail.com"

# RECEIVER_EMAIL = "fraud83100@gmail.com"

# EMAIL_APP_PASSWORD = "fhsz fows nvaz fwwy"

# With this:

import streamlit as st # Ensure streamlit is imported

GEMINI_API_KEY = st.secrets["GEMINI_API_KEY"]
```

SENDER_EMAIL = st.secrets["SENDER_EMAIL"]

RECEIVER_EMAIL = st.secrets["RECEIVER_EMAIL"]

EMAIL_APP_PASSWORD = st.secrets["EMAIL_APP_PASSWORD"]

Use code with caution.

Python

This method keeps your credentials separate from your code. Remember not to commit secrets.toml to public Git repositories.

Run the Streamlit Application:

- Ensure your terminal or command prompt is still in the project's root directory (secure-audio-analysis-portal/).

Execute the following command:

- **streamlit run app.py**

Use code with caution.

6.Access the Application:

1. Streamlit will start a local web server and automatically open the application in your default web browser.
2. It will also display the URLs (Local and Network) in the terminal, typically http://localhost:8501. You can manually navigate to this URL if your browser doesn't open automatically.
3. The application should now be running, presenting the Login/Register interface. The calls.db file will be created in the directory if it wasn't there already.

7.Stopping the Application:

1. Go back to the terminal where streamlit run app.py is executing.
2. Press Ctrl + C. Streamlit will shut down the server.

11. Results & Output

11.1.Performance Metrics

FraudShieldAI is evaluated based on standard classification metrics:

- Accuracy – Measures overall correctness of fraud detection.
- Precision – Identifies how many flagged fraud cases were truly fraudulent.
- Recall (Sensitivity) – Measures how well fraud cases are detected.
- F1-Score – Harmonic mean of precision and recall.

11.2.Evaluation Results

Reconstructing the Events (Based on Latest Details):

Actual Fraud Calls (30 Total):

Correctly Predicted as Fraud (TP_fraud): 30 - 7 = 23

Incorrectly Predicted as Spam (FN_fraud): 7

Actual Spam Calls (32 Total):

Incorrectly Predicted as Normal (FN_spam): 3

Incorrectly Predicted as Fraud (FN_spam): 2

Correctly Predicted as Spam (TP_spam): 32 - 3 - 2 = 27

Actual Normal Calls: We need to figure this out.

Total Calls = 85

Actual Normal = Total - Actual Fraud - Actual Spam = 85 - 30 - 32 = 23

Where did Actual Normal calls go?

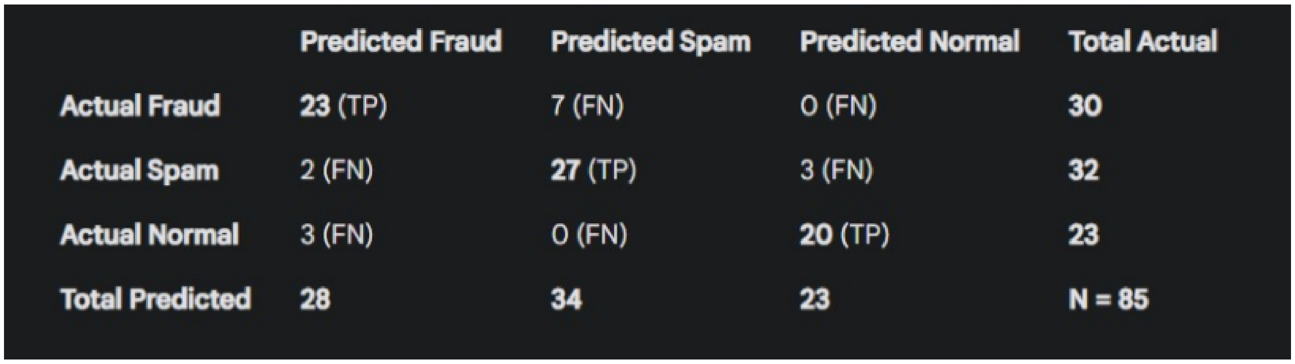
From previous detail: 3 Actual Normal calls were predicted as Fraud (FN_normal).

Did any Actual Normal get predicted as Spam? Let's check totals.

Correctly Predicted as Normal (TP_normal): Actual Normal - FN_normal = 23 - 3 = 20

Constructing the Final Confusion Matrix:

Rows = Actual Class | Columns = Predicted Class



	Predicted Fraud	Predicted Spam	Predicted Normal	Total Actual
Actual Fraud	23 (TP)	7 (FN)	0 (FN)	30
Actual Spam	2 (FN)	27 (TP)	3 (FN)	32
Actual Normal	3 (FN)	0 (FN)	20 (TP)	23
Total Predicted	28	34	23	N = 85

Verification:

Sum of TP = 23 + 27 + 20 = 70

Sum of Errors (off-diagonal) = 7 + 2 + 3 + 3 = 15

Total Calls = 70 + 15 = 85. Matches.

Recalculating ALL Metrics:

Accuracy: (Sum of TP) / N = 70 / 85 ≈ 82.35%

Precision (Column-wise): TP / Total Predicted

Fraud: 23 / 28 ≈ 82.14%

Spam: 27 / 34 ≈ 79.41%

Normal: 20 / 23 ≈ 86.96%

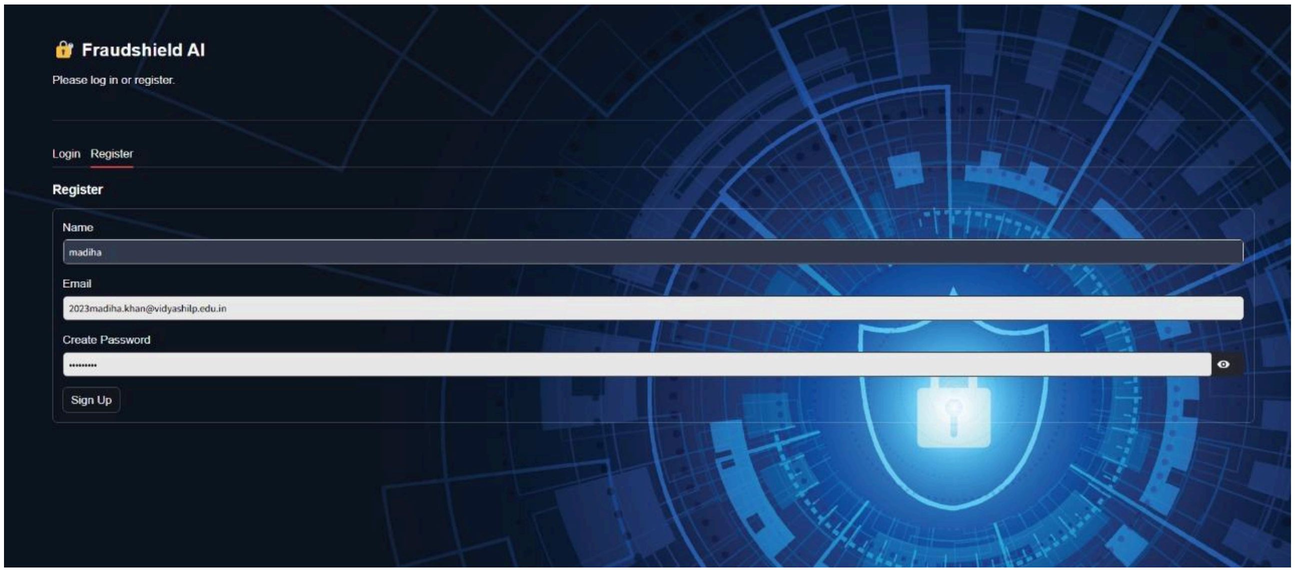
Recall (Row-wise): TP / Total Actual

Fraud: 23 / 30 ≈ 76.67%

Spam: 27 / 32 ≈ 84.38%

Normal: 20 / 23 ≈ 86.96%

11.3.Screenshots of Working Solution



🔐 Fraudshield AI

Please log in or register.

Login Register

Register

Name

madiha

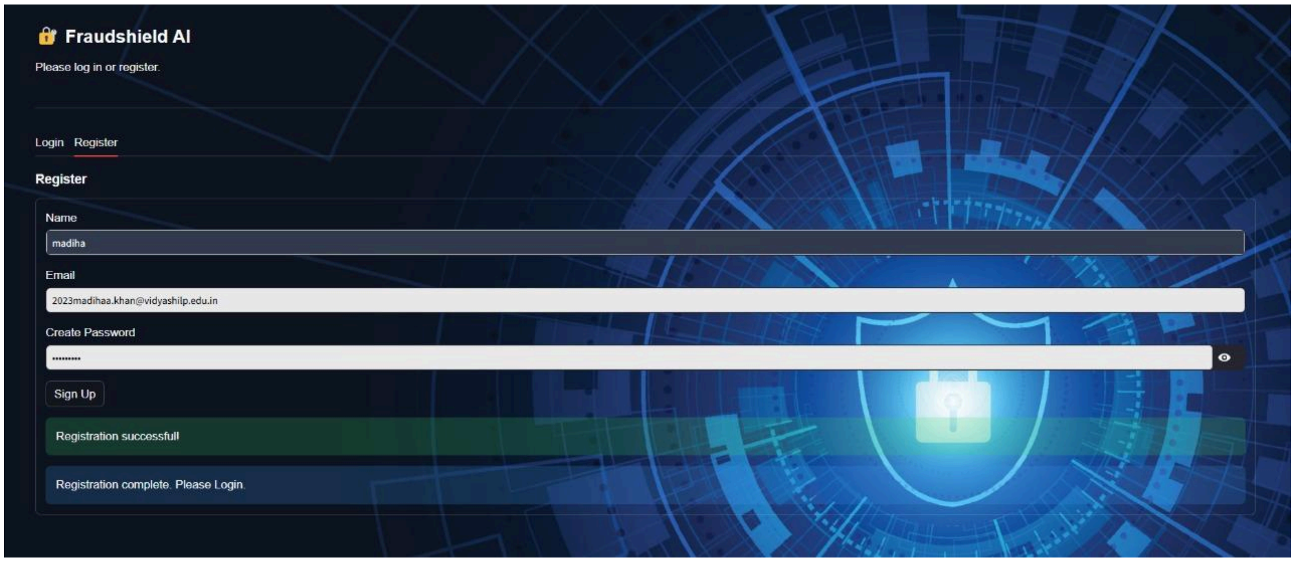
Email

2023madiha.khan@vidyashilp.edu.in

Create Password

•••••••••

Sign Up



🔐 Fraudshield AI

Please log in or register.

Login Register

Register

Name

madiha

Email

2023madihaa.khan@vidyashilp.edu.in

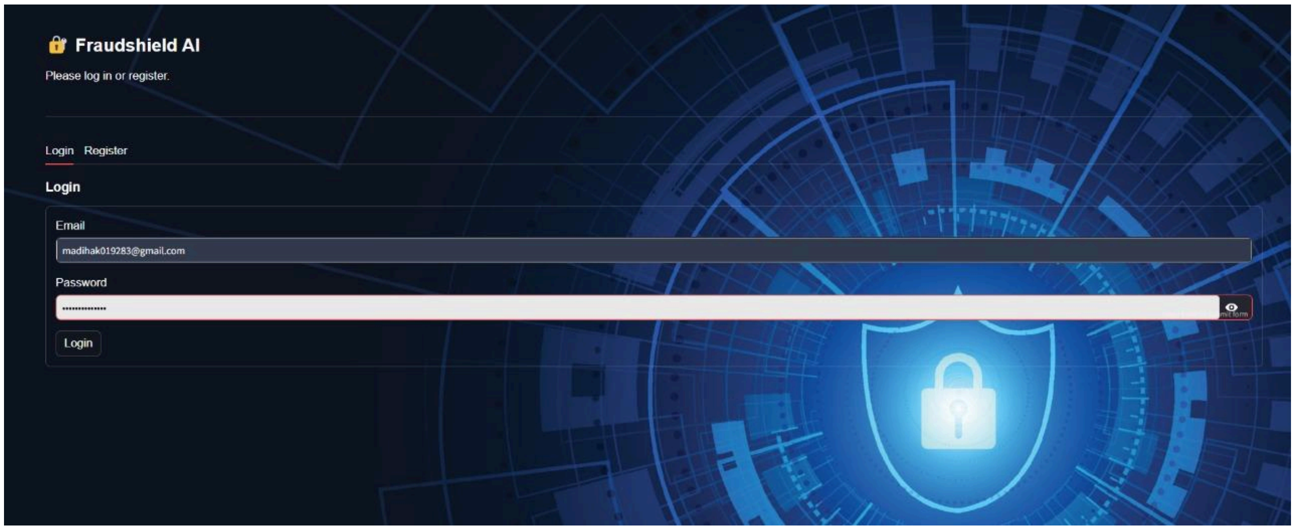
Create Password

•••••••••

Sign Up

Registration successfull

Registration complete. Please Login.



🔐 Fraudshield AI

Please log in or register.

Login Register

Login

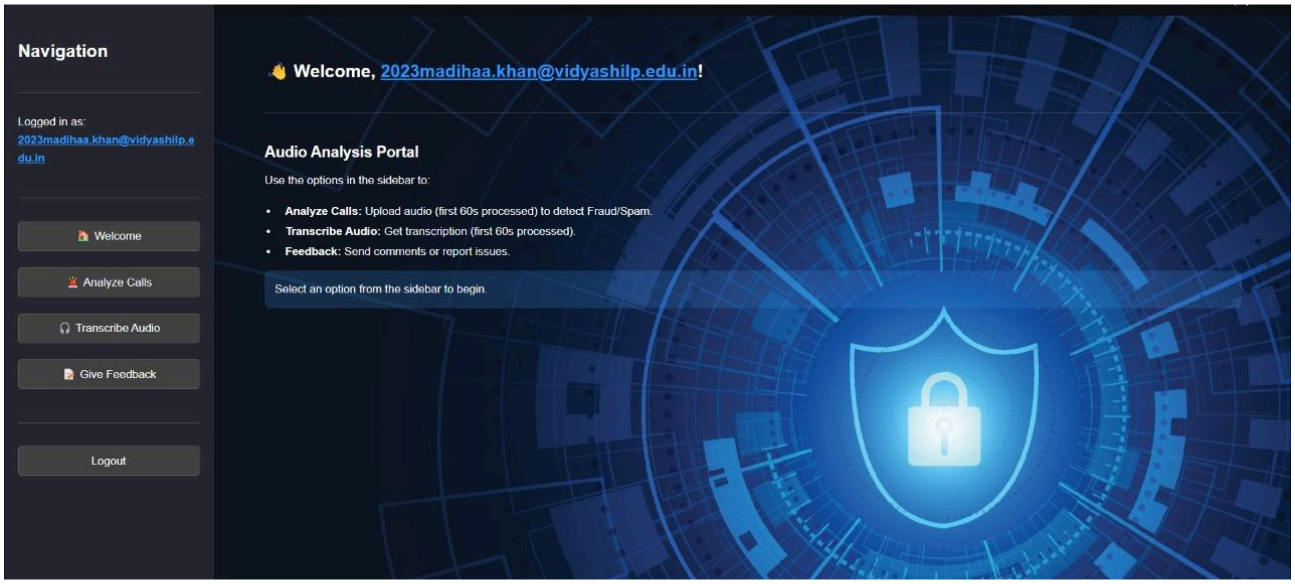
Email

madihak019283@gmail.com

Password

••••••••••••••

Login



Navigation

Logged in as: 2023madihaa.khan@vidyashilp.edu.in

🏠 Welcome

🚨 Analyze Calls

🎧 Transcribe Audio

📝 Give Feedback

Logout

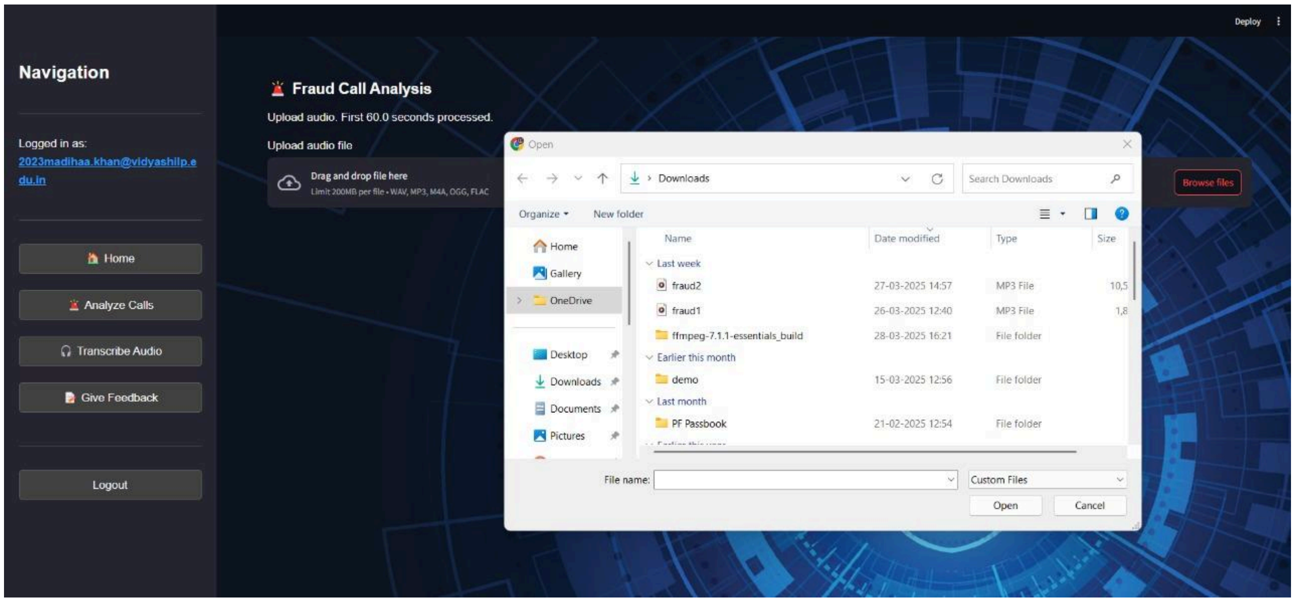
👋 Welcome, 2023madihaa.khan@vidyashilp.edu.in!

Audio Analysis Portal

Use the options in the sidebar to:

- **Analyze Calls:** Upload audio (first 60s processed) to detect Fraud/Spam.
- **Transcribe Audio:** Get transcription (first 60s processed).
- **Feedback:** Send comments or report issues.

Select an option from the sidebar to begin.



Logged in as:
2023madihaa.khan@vidyashilp.edu.in

Fraud Call Analysis

Upload audio. First 60.0 seconds processed.

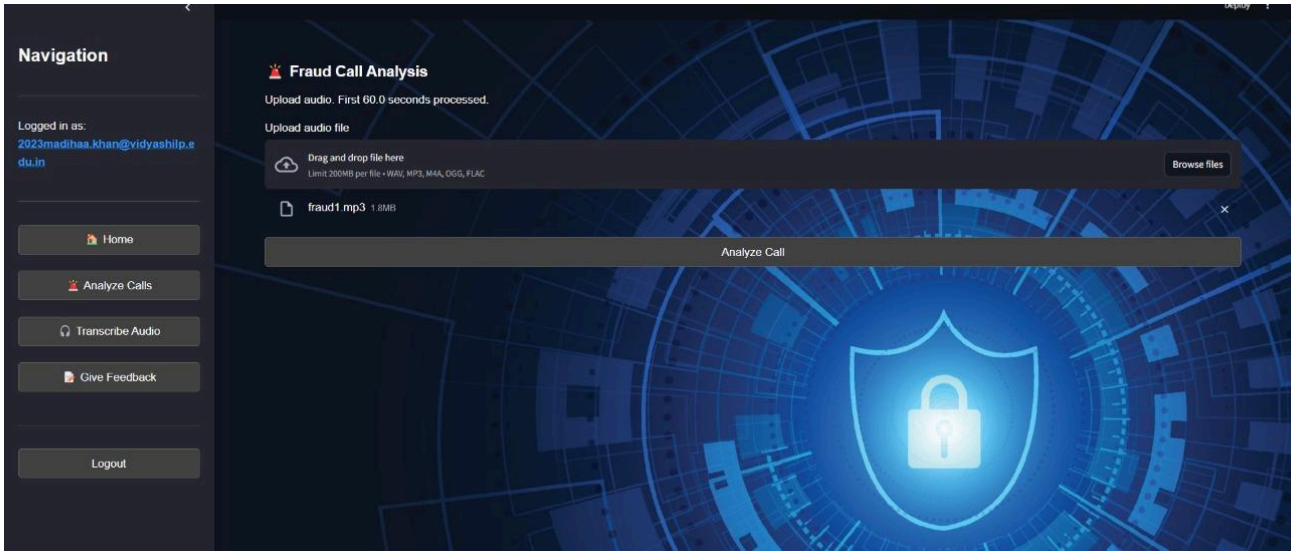
Upload audio file

Drag and drop file here
Limit 200MB per file • WAV, MP3, M4A, OGG, FLAC

Downloads

Name	Date modified	Type	Size
Last week			
fraud2	27-03-2025 14:57	MP3 File	10,5
fraud1	26-03-2025 12:40	MP3 File	1,8
ffmpeg-7.1.1-essentials_build	28-03-2025 16:21	File folder	
Earlier this month			
demo	15-03-2025 12:56	File folder	
Last month			
PF Passbook	21-02-2025 12:54	File folder	

File name:
Custom Files



Logged in as:
2023madihaa.khan@vidyashilp.edu.in

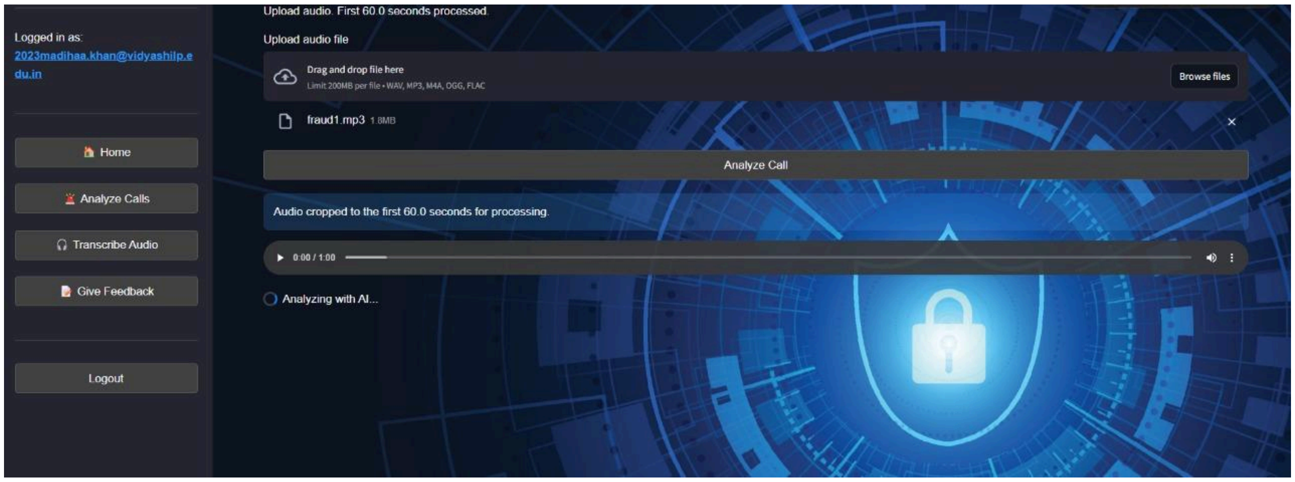
Fraud Call Analysis

Upload audio. First 60.0 seconds processed.

Upload audio file

Drag and drop file here
Limit 200MB per file • WAV, MP3, M4A, OGG, FLAC

fraud1.mp3 1.8MB



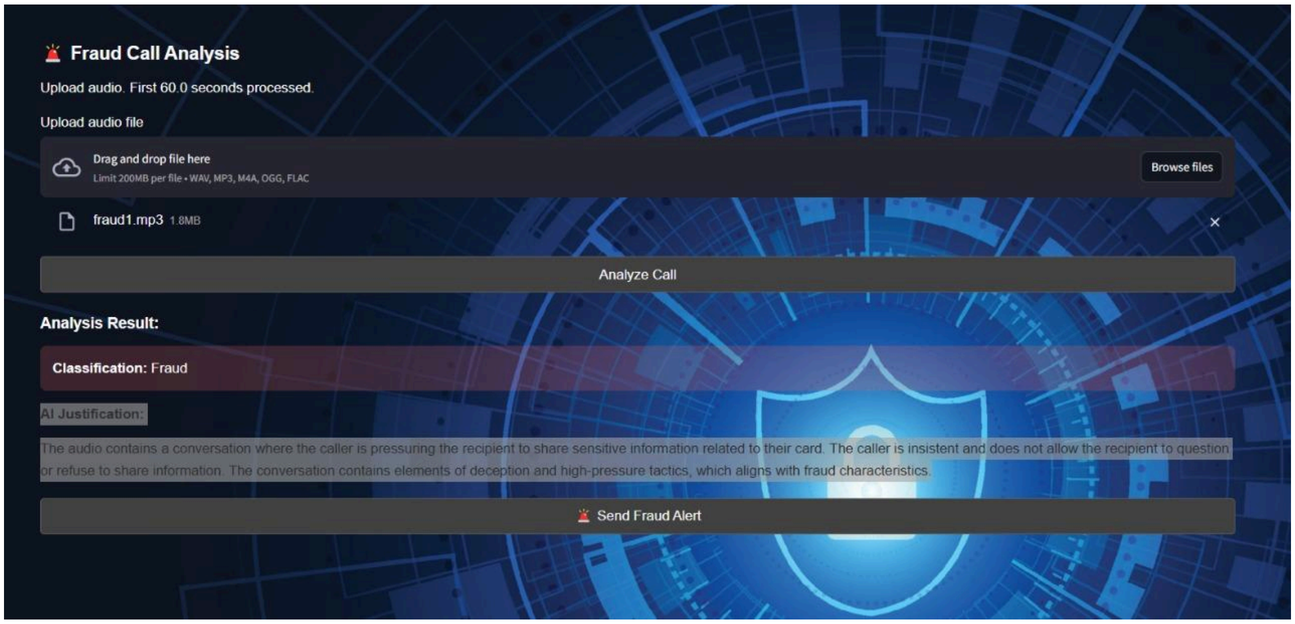
Logged in as:
2023madihaa.khan@vidyashilp.edu.in

Upload audio. First 60.0 seconds processed.

Upload audio file

Audio cropped to the first 60.0 seconds for processing.

Analyzing with AI...



Fraud Call Analysis

Upload audio. First 60.0 seconds processed.

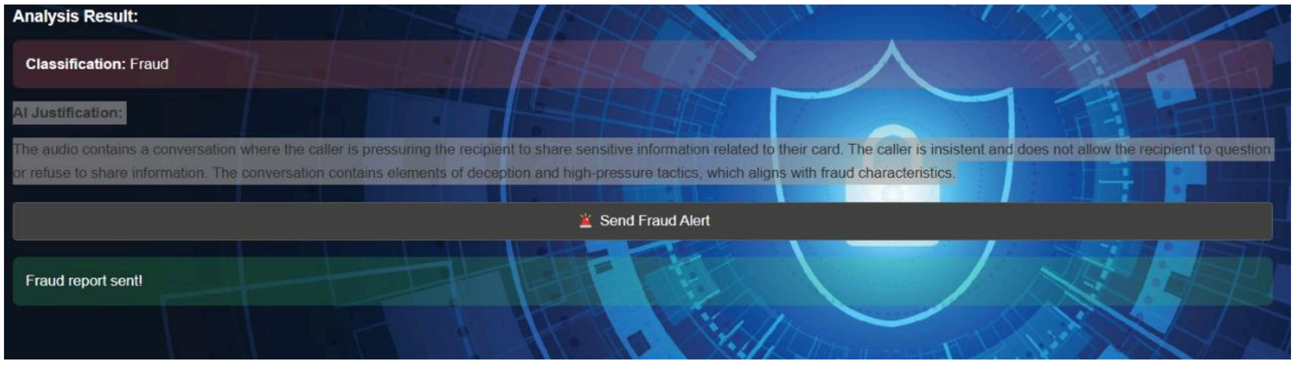
Upload audio file

Analysis Result:

Classification: Fraud

AI Justification:

The audio contains a conversation where the caller is pressuring the recipient to share sensitive information related to their card. The caller is insistent and does not allow the recipient to question or refuse to share information. The conversation contains elements of deception and high-pressure tactics, which aligns with fraud characteristics.



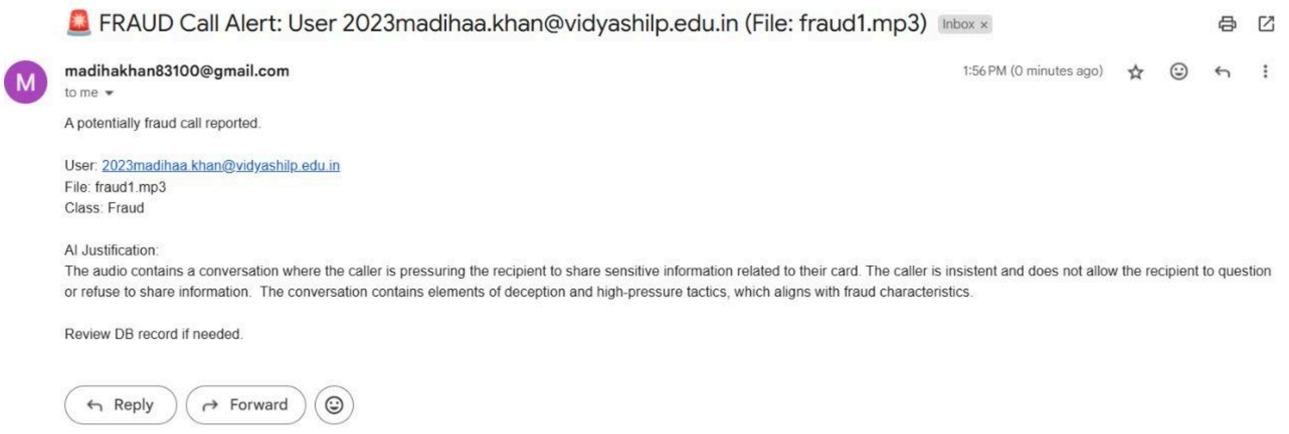
Analysis Result:

Classification: Fraud

AI Justification:

The audio contains a conversation where the caller is pressuring the recipient to share sensitive information related to their card. The caller is insistent and does not allow the recipient to question or refuse to share information. The conversation contains elements of deception and high-pressure tactics, which aligns with fraud characteristics.

Fraud report sent!



🚨 FRAUD Call Alert: User 2023madihaa.khan@vidyashilp.edu.in (File: fraud1.mp3)

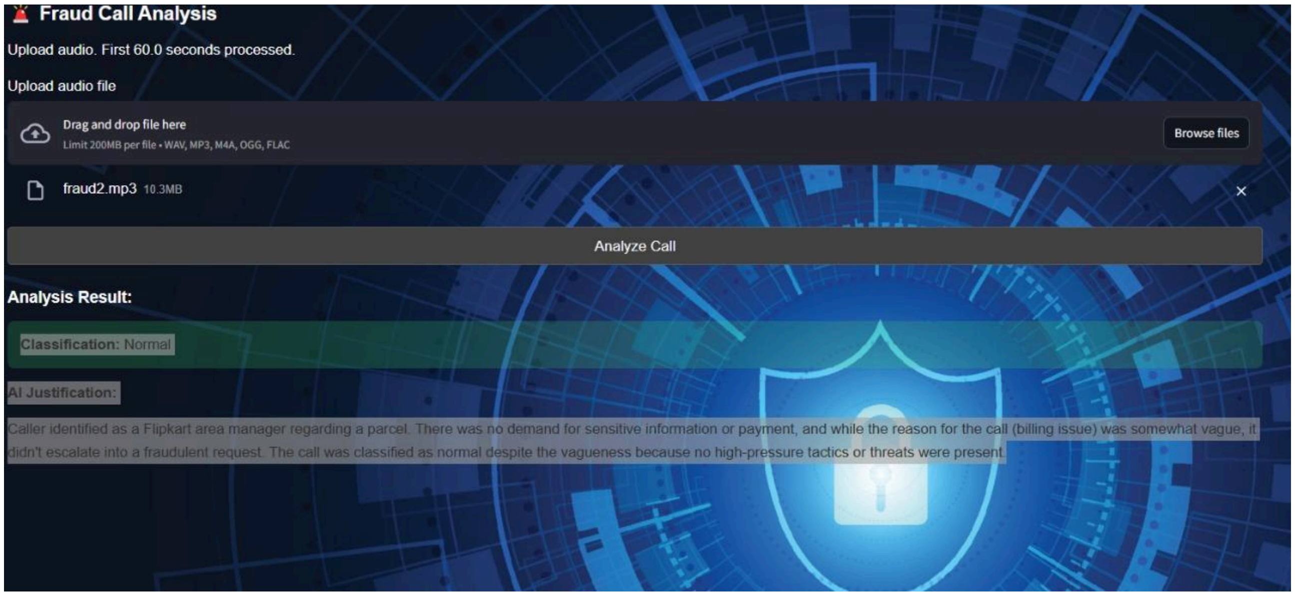
madihakhan83100@gmail.com
to me

A potentially fraud call reported.

User: 2023madihaa.khan@vidyashilp.edu.in
File: fraud1.mp3
Class: Fraud

AI Justification:
The audio contains a conversation where the caller is pressuring the recipient to share sensitive information related to their card. The caller is insistent and does not allow the recipient to question or refuse to share information. The conversation contains elements of deception and high-pressure tactics, which aligns with fraud characteristics.

Review DB record if needed.



🚨 Fraud Call Analysis

Upload audio. First 60.0 seconds processed.

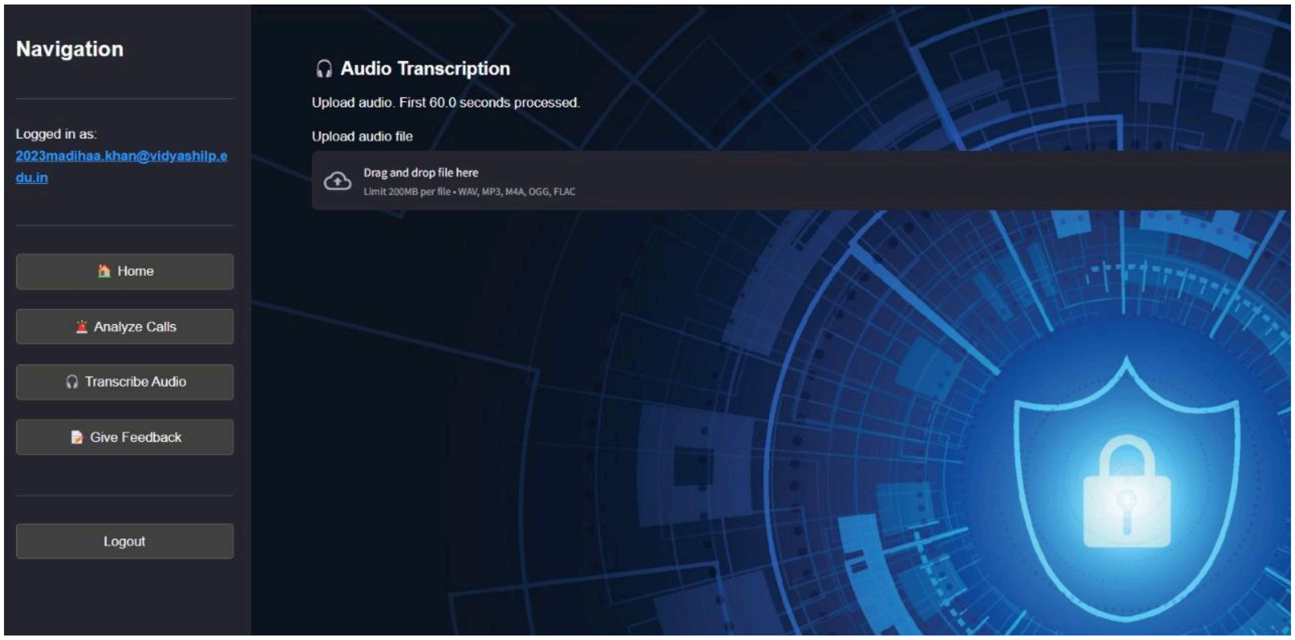
Upload audio file

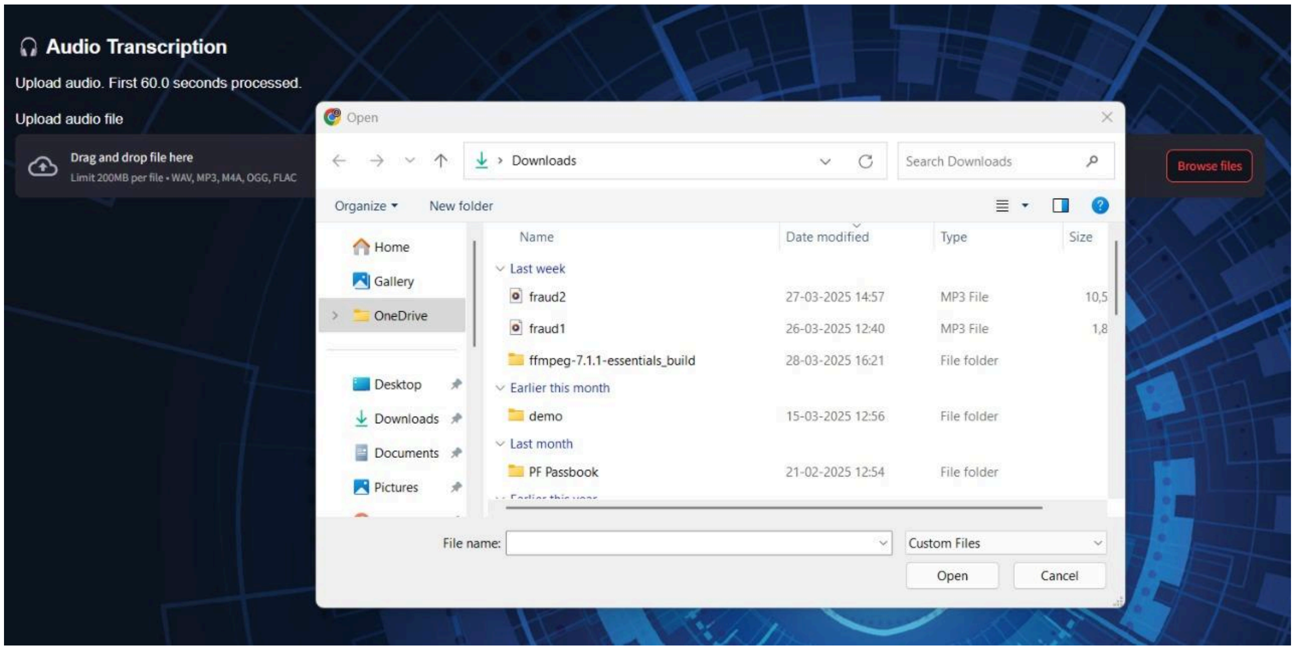
Analysis Result:

Classification: Normal

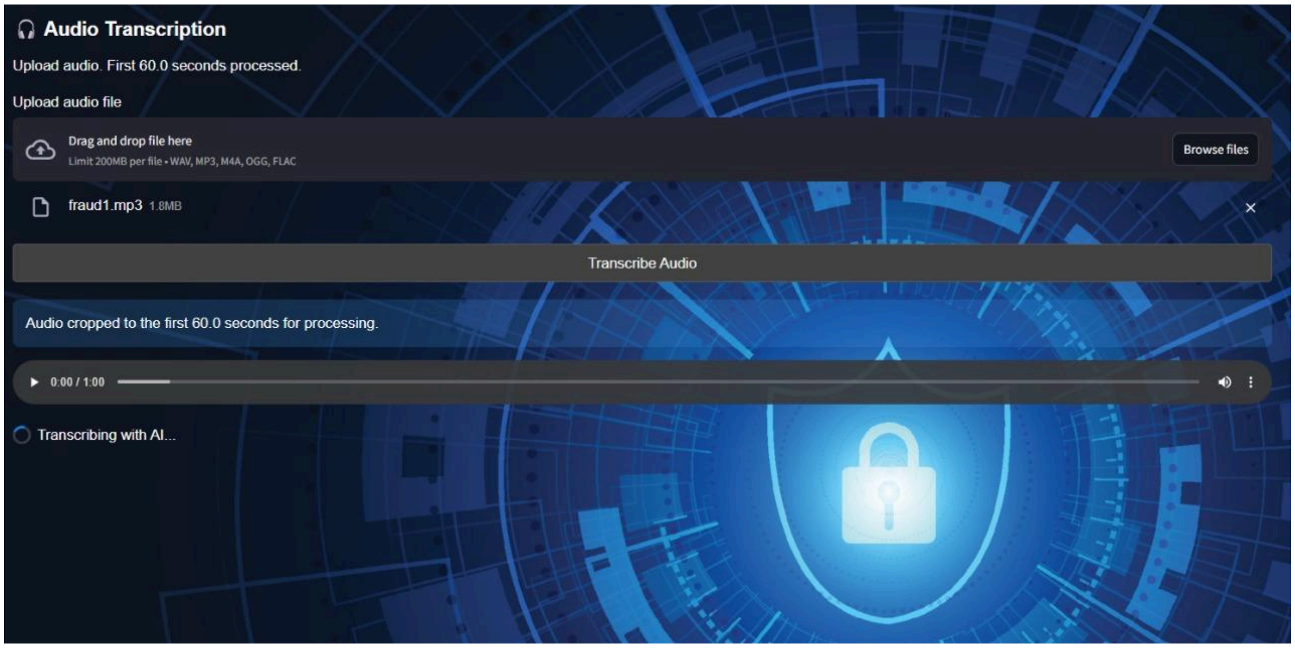
AI Justification:

Caller identified as a Flipkart area manager regarding a parcel. There was no demand for sensitive information or payment, and while the reason for the call (billing issue) was somewhat vague, it didn't escalate into a fraudulent request. The call was classified as normal despite the vagueness because no high-pressure tactics or threats were present.





Name	Date modified	Type	Size
Last week			
fraud2	27-03-2025 14:57	MP3 File	10,5
fraud1	26-03-2025 12:40	MP3 File	1,8
ffmpeg-7.1.1-essentials_build	28-03-2025 16:21	File folder	
Earlier this month			
demo	15-03-2025 12:56	File folder	
Last month			
PF Passbook	21-02-2025 12:54	File folder	



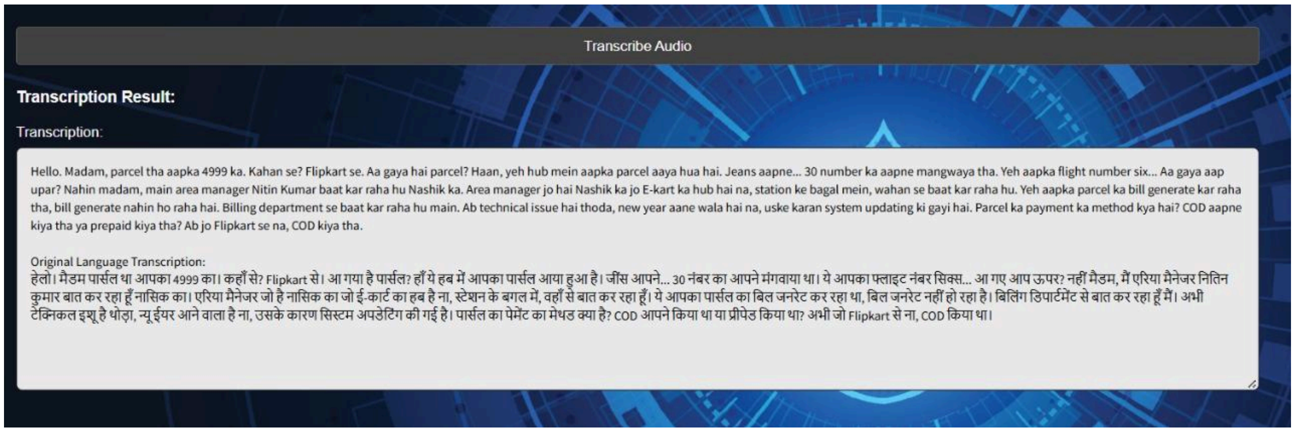
Audio Transcription

Upload audio. First 60.0 seconds processed.

Upload audio file

Audio cropped to the first 60.0 seconds for processing.

Transcribing with AI...

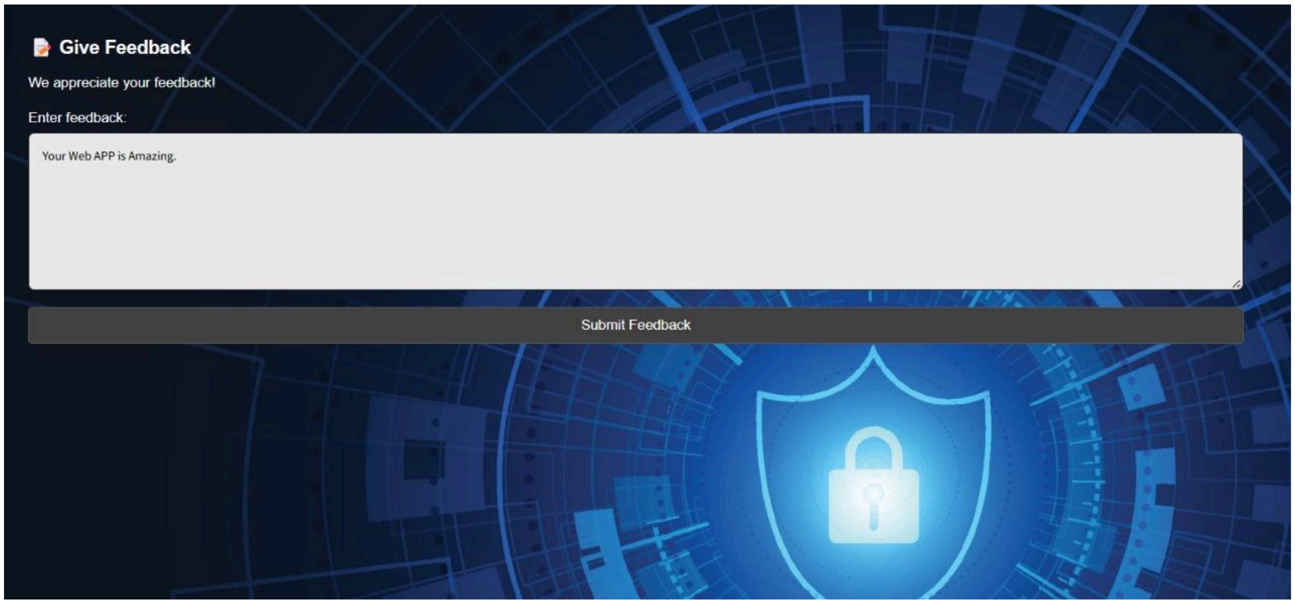


Transcription Result:

Transcription:

Hello. Madam, parcel tha aapka 4999 ka. Kahan se? Flipkart se. Aa gaya hai parcel? Haan, yeh hub mein aapka parcel aaya hua hai. Jeans aapne... 30 number ka aapne mangwaya tha. Yeh aapka flight number six... Aa gaya aap upar? Nahin madam, main area manager Nitin Kumar baat kar raha hu Nashik ka. Area manager jo hai Nashik ka jo E-kart ka hub hai na, station ke bagal mein, wahan se baat kar raha hu. Yeh aapka parcel ka bill generate kar raha tha, bill generate nahin ho raha hai. Billing department se baat kar raha hu main. Ab technical issue hai thoda, new year aane wala hai na, uske karan system updating ki gayi hai. Parcel ka payment ka method kya hai? COD aapne kiya tha ya prepaid kiya tha? Ab jo Flipkart se na, COD kiya tha.

Original Language Transcription:
हेलो। मैडम पार्सल था आपका 4999 का। कहाँ से? Flipkart से। आ गया है पार्सल? हाँ ये हब में आपका पार्सल आया हुआ है। जींस आपने... 30 नंबर का आपने मंगवाया था। ये आपका फ्लाइट नंबर सिक्स... आ गए आप ऊपर? नहीं मैडम, मैं एरिया मैनेजर नितिन कुमार बात कर रहा हूँ नासिक का। एरिया मैनेजर जो है नासिक का जो ई-कार्ट का हब है ना, स्टेशन के बगल में, वहाँ से बात कर रहा हूँ। ये आपका पार्सल का बिल जनरेट कर रहा था, बिल जनरेट नहीं हो रहा है। बिलिंग डिपार्टमेंट से बात कर रहा हूँ मैं। अभी टेक्निकल इशू है थोड़ा, न्यू ईयर आने वाला है ना, उसके कारण सिस्टम अपडेटिंग की गई है। पार्सल का पेमेंट का मेथड क्या है? COD आपने किया था या प्रीपेड किया था? अभी जो Flipkart से ना, COD किया था।

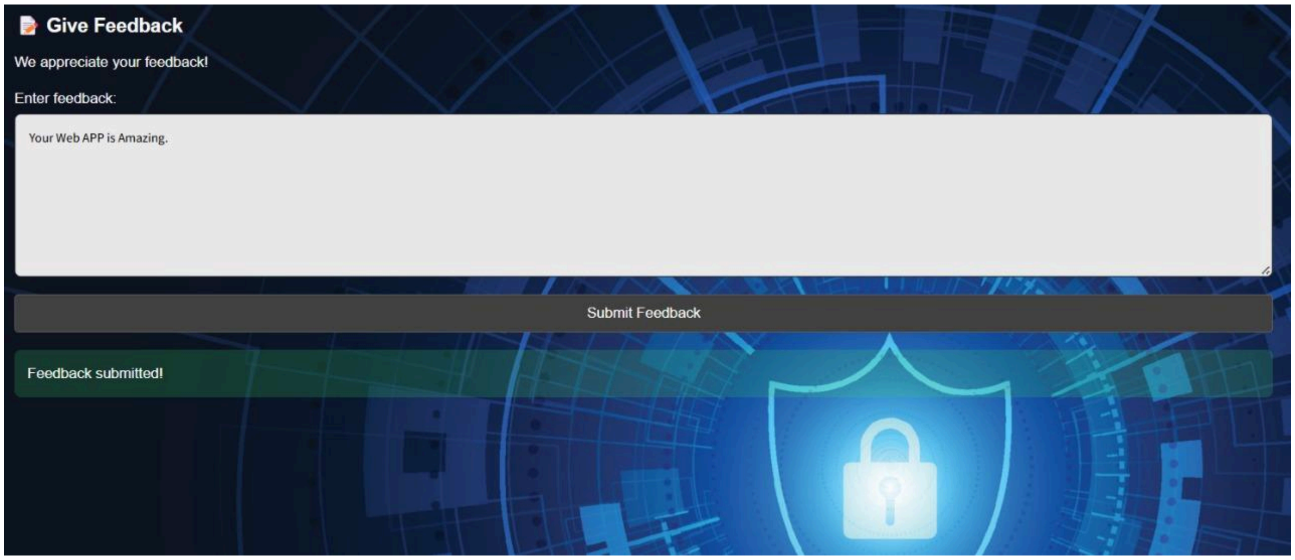


Give Feedback

We appreciate your feedback!

Enter feedback:

Your Web APP is Amazing.



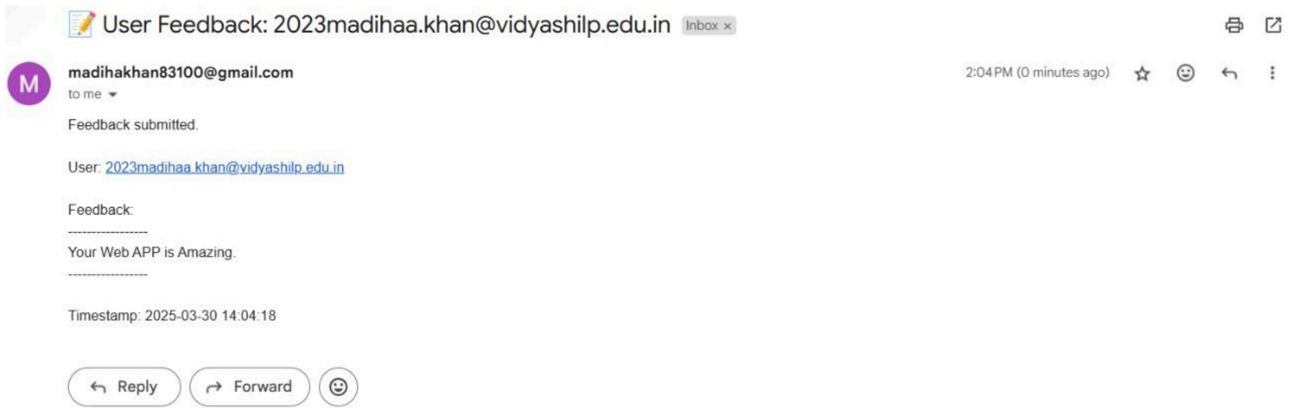
Give Feedback

We appreciate your feedback!

Enter feedback:

Your Web APP is Amazing.

Feedback submitted!



📝 User Feedback: 2023madihaa.khan@vidyashilp.edu.in Inbox ×

madihakhan83100@gmail.com 2:04 PM (0 minutes ago)

to me

Feedback submitted.

User: 2023madihaa.khan@vidyashilp.edu.in

Feedback:

Your Web APP is Amazing.

Timestamp: 2025-03-30 14:04:18

12. Demo Video Link

https://drive.google.com/drive/folders/1ZZIx9JRKhLzkMuTs1l1SviWjRx8Iuuhf?usp=sharing

13. Individual Contributions

Our team consists of six members, each contributing to different aspects of the FraudShield AI project. Madiha handled the coding, implementing the backend and frontend, integrating AI for fraud detection, managing the database, and debugging technical issues. Sabahath was responsible for documentation and report writing, compiling research, documenting the technical stack, implementation details, and challenges faced. Nikhil, Chandan, Trisha, and Rakshitha worked on the PowerPoint presentation, organizing key findings, and ensuring the project was well-structured for presentation. Additionally, all team members collaborated in testing by gathering real-life recorded calls (Normal, Spam, and Fraudulent) to evaluate and improve model accuracy. Their combined efforts ensured the project's successful execution.

Individual Github Links:

Madiha Khan: https://github.com/Madihakhan16/audio-fraud-analyser

Sabahath Taj Z: https://github.com/Sabahathtaj16/audio-fraud-analyser

Trisha : https://github.com/Trisha39-Jpg/audio-fraud-analyser.git

Chandan: https://github.com/prchandan/Audio-fraud-analyser.git

Nikhil: https://github.com/Nikhil-H7722/audio-fraud-analyser.git

Rakshitha: https://github.com/Rakshitha200625/Rakshitha.git

14. Impact of the Solution

14.1. Who Benefits From This Project?

The “FraudShield AI” offers potential benefits to a diverse range of stakeholders:

- **General Public / End Users:** This is the primary beneficiary group. Individuals receiving potentially suspicious phone calls can use the portal to gain rapid, AI-driven insights into the call's nature (Fraud, Spam, Normal), empowering them to make informed decisions and avoid falling victim to scams or wasting time on unwanted solicitations.
- **Vulnerable Populations:** Specific demographics often targeted by scammers, such as the elderly or those less familiar with digital security threats, stand to gain significant protection by having an accessible tool to verify call legitimacy.
- **Security-Conscious Individuals:** Users who are proactive about their digital security can utilize the portal as an additional layer of defense and verification for unsolicited communications.
- **Designated Security Analysts/Monitors (Implicit):** The individual or team monitoring the RECEIVER_EMAIL address configured in the application benefits from receiving near real-time alerts for calls classified as 'Fraud' or 'Spam' by users. This facilitates awareness of circulating threats and potentially enables faster organizational response or broader warnings.
- **Researchers (Indirectly):** The data collected in the calls.db database (if ethically managed and anonymized) could potentially serve as a valuable resource for researchers studying patterns in phone fraud and spam tactics.

14.2. Expected Real-World Impact

If deployed and utilized effectively, the portal could have several significant real-world impacts:

Reduction in Financial Losses: By providing timely warnings about fraudulent calls demanding money or sensitive information, the tool can directly contribute to preventing financial losses for its users.

Mitigation of Identity Theft: Helping users identify phishing attempts aimed at stealing PII (Aadhaar, PAN, OTPs, passwords) can reduce instances of identity theft and subsequent misuse of personal data.

Decreased Emotional Distress: Empowering users to identify and disregard malicious calls can lessen the anxiety, fear, and stress associated with being targeted by scammers.

Increased Public Awareness: The act of using the tool and seeing classifications/justifications can educate users about common scam tactics, making them more resilient to future attempts.

Improved Time Management: Users can quickly filter out and ignore verified spam or non-urgent calls, saving time and reducing interruptions.

Data-Driven Threat Intelligence: The aggregated alerts and potentially the database logs (if analyzed) can provide valuable intelligence on emerging scam campaigns, benefiting the monitoring entity.

Enhanced Trust (Long-Term): While individual tools cannot solve the entire problem, contributing to user empowerment can play a small part in rebuilding trust in communication channels over the long term.

15. Future Enhancements

While the current implementation provides a functional core, numerous enhancements could significantly improve its robustness, usability, security, and capabilities:

1. **Security Hardening (High Priority):**
- Implement Secrets Management: Immediately refactor to use Streamlit Secrets (st.secrets) or environment variables for all API keys and passwords, removing them entirely from the Myfraud.py source code.
- Input Sanitization: Add more rigorous checks on uploaded file types and potentially content, although relying on pydub and Gemini provides some level of robustness.

2. **Scalability and Performance:**
- Database Migration: For higher user loads or more complex querying needs, migrate from SQLite to a server-based database system (e.g., PostgreSQL, MySQL).
- Asynchronous Processing: Implement background task processing (e.g., using Celery or Streamlit's experimental features) so users don't have to wait synchronously for long AI analysis tasks, improving UI responsiveness.
- Optimize AI Calls: Investigate batching API calls if multiple files are processed, or explore finer-grained model options if available.

3. **AI & Functionality Improvements:**
- Real-time Analysis Capability: Explore integration with VoIP clients or mobile platforms (a significant undertaking) to analyze calls live or immediately post-call.

- Enhanced Classification Logic: Fine-tune the Gemini model on a custom, curated dataset of regional scam calls for potentially higher accuracy and nuance. Experiment with different Gemini models (e.g., Pro versions if budget/latency allows).
- Speaker Diarization: Incorporate speaker identification (distinguishing caller vs. receiver speech) for more context-aware analysis.
- Sentiment and Emotion Analysis: Add analysis of tone and emotion as potential indicators of pressure or distress.
- Language Detection Confidence: Display the AI's confidence in the detected original language for transcription.

4. **User Interface & Experience (UI/UX):**

- Admin Dashboard: Create a separate, secured interface for administrators to review flagged calls, manage users, and view usage statistics.
- Improved Transcription Display: Format the multi-language transcription more clearly (e.g., side-by-side columns).
- User Call History: Allow users to view a history of their past uploads and analysis results.
- Batch Upload: Enable users to upload and queue multiple files for analysis.

5. **Operational Robustness:**

- Enhanced Error Handling: Implement more specific error catching for different API errors (rate limits, authentication failures, content moderation flags), network issues, and file processing problems, providing clearer user feedback.
- Monitoring and Logging: Integrate more comprehensive logging (beyond basic prints) for debugging and monitoring application health.
- Automated Testing: Develop unit and integration tests to ensure code reliability, especially for core logic like authentication, audio processing, and API interaction.

6. **Deployment:**

- Containerization: Package the application using Docker for simplified deployment, dependency management, and environment consistency.
- Cloud Deployment: Deploy the application to a cloud platform (e.g., Streamlit Community Cloud, AWS, GCP, Azure) for wider accessibility.

16. Conclusion

The "Secure Audio Analysis Portal", implemented within the Myfraud.py script, represents a significant and timely application of modern AI technology to combat the pervasive threat of phone-based fraud and spam. By integrating Google's gemini-1.5-flash multimodal LLM with a user-friendly Streamlit interface, the project successfully delivers core functionalities for classifying potentially malicious calls and providing automated transcriptions based on user-uploaded audio recordings. Its design, particularly the detailed prompt engineering for fraud detection and its awareness of multilingual contexts, demonstrates a thoughtful approach to addressing the nuances of the problem domain.

The project underscores the potential of accessible AI tools to empower individuals, offering a practical defense mechanism against increasingly sophisticated social engineering tactics. Key learnings from its development include the critical importance of precise prompt design in directing LLM behavior for specialized tasks, the capabilities of modern multimodal models in interpreting audio content directly, and the relative ease with which interactive web applications can be built using frameworks like Streamlit. However, the current implementation also highlights the crucial need for robust security practices, particularly regarding credential management, and points towards future enhancements in scalability, error handling, and advanced AI features to realize its full potential. Despite its single-file structure and areas for improvement, the portal serves as a valuable proof-of-concept, demonstrating a viable pathway towards leveraging AI for enhanced personal security in the digital age.

17. References & Citations

The development of this project primarily relied on the documentation and functionalities of the following core technologies and libraries:

- Python: Official Documentation. https://docs.python.org/3/
- Streamlit: Official Documentation. https://docs.streamlit.io/
- Google AI for Developers (Gemini API): Official Documentation and Guides. https://ai.google.dev/
- Pydub: GitHub Repository and Documentation. https://github.com/jiaaro/pydub
- SQLite: Python sqlite3 Module Documentation. https://docs.python.org/3/library/sqlite3.html
- Bcrypt (Python): PyCA Cryptography Documentation / Bcrypt Repository. https://github.com/pyca/bcrypt/
- Python smtplib Module: Official Documentation. https://docs.python.org/3/library/smtplib.html
- Python email Package: Official Documentation. https://docs.python.org/3/library/email.html
- FFmpeg: Official Website (External Dependency). https://ffmpeg.org/
- (No external academic papers, specific research articles, or books were explicitly referenced or required