EchoGuard: Project Documentation

Project Title: Audio and AI-Based Movie Genre and Rating Detector

Team Members: Abner Rodriguez (*PID 6401948*), Alfonso Contreras (*PID 6456403*), Kevin Sanchez (*PID 6416454*), Carlos Navarro (*PID 6194280*), Kenneth Martin (*PID 6304581*)

Executive Summary

Thanks to websites like YouTube, TikTok, and streaming services, kids are now exposed to more digital entertainment than ever before. While the majority of current parental control systems focus on visual cues and content ratings, they often overlook the audio environment, where irritating background noise, inappropriate speech, or dangerous sounds may go unnoticed. Particularly when the audio content is fast-paced, blended, or integrated into other media, parents do not have an easy means to evaluate what their kids are hearing.

By offering a sophisticated audio-safety analysis tool created especially for families, EchoGuard fills this gap. Users can record or upload audio directly from their browsers, and the technology provides clear, instantaneous insights into both speech and non-speech sounds. EchoGuard uses a Flask backend to process each audio segment through a number of steps, such as voice activity detection, non-speech validation, full-clip transcription using Faster-Whisper, environmental sound classification using a PyTorch CNN, and a final safety assessment driven by Google's Gemini 2.5 Pro model. As a result, parents can rapidly assess whether audio content is safe or worrying thanks to a uniform risk and benefit summary.

EchoGuard ultimately serves as a supportive tool for families, enabling parents to stay informed about their children's audio environments and make safer, more confident decisions about the media they consume.

Problem & Requirements

Stakeholders & Personas

Primary Stakeholders:

1. Parents & Guardians
- Individuals responsible for monitoring children's media exposure.
2. Children (End Beneficiaries)
- The individuals whose safety and well-being the system aims to protect.
3. Educators
- Professionals who may use the instrument to assess classroom or digital learning audio

Personas:

1. Busy Parent
- A parent who lacks the time to physically listen to the complete audio track but wants to quickly determine whether a video or recording is appropriate.
2. Babysitter
- A babysitter who needs to ensure the kids they are babysitting are not consuming dangerous media.

Functional Requirements

- Users must be able to capture or submit audio straight from the browser.
- For real-time feedback, audio must be processed by the backend in 5-second intervals.

- To distinguish between speech and non-speech, the system must use voice activity detection (VAD).
- Validation heuristics must be used to assess non-speech audio.
- A CNN environmental sound model must be used to categorize validated audio.
- Faster-Whisper must be used by the system to generate a full-clip transcription.
- Gemini 2.5 Pro must be used by the system to produce a risk and benefit analysis.
- The final safety score and real-time sound detection findings must be displayed on the frontend.

Non-Functional Requirements

- The interface must be straightforward, easily navigable, and accessible to non-technical users.
- Low-latency processing (<1 second per clip) must be provided by the system.
- For complete recordings, transcription must be finished in a fair amount of time.
- After the session is over, the audio must be momentarily saved and then erased.
- Raw recordings are not kept for an extended period of time.
- Noisy situations and fluctuating audio quality must be handled by the system.

Constraints & Success Criteria

Constraints:

- Restrictions of the browser microphone API.
- On-demand CNN inference and VAD performance limitations.
- Variability in user audio (different devices, quiet recordings, background noise).

- restricted environmental sound class dataset.

Success Criteria:

- Precise categorization of typical environmental sound types.
- Full, high-quality speech transcription.
- Safety scoring that is easy for parents to comprehend.
- Live sound detection with low latency.

Risks

- Model Bias: Rare or unclear noises may be misinterpreted by the CNN.
- False Negatives: Potentially dangerous noises may be missed.
- Privacy Concerns: Users could worry about recorded audio being misused, although this is lessened by the in-memory storage policy.
- Overreliance: Parents may rely on the system without keeping an eye on their kids' material.
- Technical Variability: VAD performance may be impacted by various microphones and browser settings.

Prioritized Backlog

1. Put in place a chunk-based upload pipeline and audio recording.
2. Separate speech/non-speech segmentation with RNNoise VAD.
3. Construct a validation module for non-speech (RMS, amplitude, duration).
4. Incorporate the CNN environmental sound classifier.
5. Use a session manager to store audio and results temporarily.

6. Create a complete transcription process with Faster-Whisper.
7. For final scoring and explanations, connect Gemini 2.5 Pro.
8. Create a frontend result display for environmental noises that is updated in real time.
9. Create a dashboard that includes the scoring and transcript.
10. Include error handling for model failures, VAD edge situations, and missing audio.
11. Add more sound classes to increase the accuracy of the model.
12. Improve the user interface for accessibility and clarity.
13. Incorporate Supabase to ensure data durability in the future.
14. Test on various devices and browsers.
15. Create the poster, documentation, and demonstration materials.

System Overview & Architecture

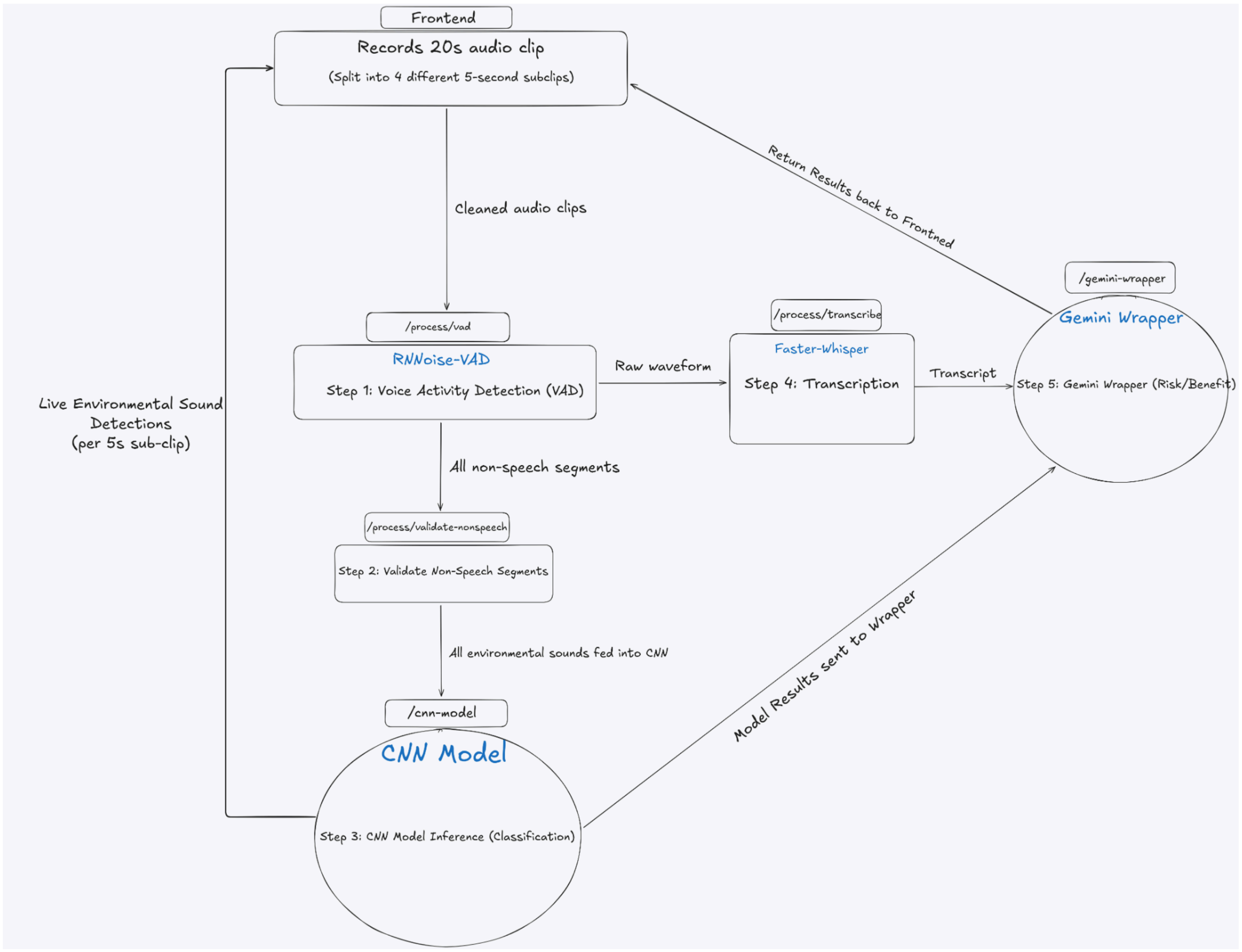
High-Level System Overview

EchoGuard is a service-oriented system that can process audio in real time and produce a comprehensive safety assessment after a full recording is finished. Live recording and file upload are the two modes of operation of the system, and both use the same backend processing pipeline.

The Flask backend, which handles vocal activity recognition, non-speech validation, environmental sound classification, transcription, and risk/benefit assessment, receives audio from the Next.js frontend in 5-second intervals. To ensure maintainability and a clear division of responsibilities, the backend is divided into specialized modules, each of which is in charge of a single stage of analysis.

In addition to providing immediate environmental sound input, the system is built to prepare the entire recording for transcription and ultimate AI-driven assessment.

Architecture Diagram


Frontend
Records 20s audio clip
(Split into 4 different 5-second subclips)
Cleaned audio clips
Return Results back to Frontned
/gemini-wrapper
Gemini Wrapper
Step 5: Gemini Wrapper (Risk/Benefit)
/process/vad
RNNoise-VAD
Step 1: Voice Activity Detection (VAD)
Raw waveform
/process/transcribe
Faster-Whisper
Step 4: Transcription
Transcript
Live Environmental Sound Detections (per 5s sub-clip)
All non-speech segments
/process/validate-nonspeech
Step 2: Validate Non-Speech Segments
All environmental sounds fed into CNN
/cnn-model
CNN Model
Step 3: CNN Model Inference (Classification)
Model Results sent to Wrapper

Technologies, Frameworks & Services

Frontend Technologies

• Next.js – User interface, audio recording, and real-time updates

• Web Audio API – Audio capture and chunking

Backend Technologies

• Flask – REST API using blueprint architecture

• RNNoise – Voice activity detection (speech vs. non-speech)

• PyTorch – Environmental sound CNN inference

• Faster-Whisper – Full-clip speech transcription

• Google Gemini 2.5 Pro – Risk/benefit reasoning and safety scoring

• Torchaudio – Resampling and waveform manipulation

• NumPy – Audio preprocessing and heuristics

API Overview

EchoGuard exposes a small set of REST endpoints organized under Flask blueprints:

1. **/process/vad**

- Handles incoming audio clips, performs normalization, VAD, speech extraction, non-speech stitching, and routes validated non-speech to subsequent stages.

2. **/process/validate-non-speech**

- Runs amplitude- and duration-based heuristics to determine whether non-speech audio is meaningful enough to classify.

3. **/cnn-model**

- Receives validated non-speech audio and returns CNN-based environmental sound predictions.

4. **/process/transcribe**

- Triggered when the final clip arrives, merges all raw audio segments and generates a complete transcript.

5. **/process/gemini**

- Combines CNN outputs and transcription into a structured rubric evaluation using Gemini 2.5 Pro.

Data & Storage Overview

EchoGuard uses a session & in-memory based model to ensure simplicity and efficiency.

Temporary In-Memory Storage (Current)

- Speech segments
- Validated non-speech results
- Raw full-clip audio
- Transcription text
- Final Gemini scoring

Discipline-Specific Depth

Algorithms & Data Structures

1. Voice Activity Detection (VAD)

- EchoGuard classifies every audio frame as either speech or non-speech using RNNoise, a recurrent neural network-based VAD method.

Important traits:

- Frame-based inference, operating at about 48 kHz
- High accuracy in noisy situations; low processing complexity appropriate for real-time use

2. Non-Speech Validation

EchoGuard uses simple statistical algorithms after VAD to make sure that only relevant ambient audio is processed:

- Amplitude threshold for Root Mean Square (RMS)
- The threshold for peak amplitude
- Loud-sample ratio (the percentage of frames that are louder than a dB threshold)

3. CNN Model

A PyTorch convolutional neural network trained on mel-spectrogram inputs is used by EchoGuard.

Considerations for complexity:

- The CNN design enables effective time-frequency pattern extraction; inference is almost real-time due to the tiny model size.
- The whole probability distribution and top-1 forecast are both produced by the model.

4. Transcription Tool (Faster-Whisper)

Sequence-to-sequence voice recognition using a modified Whisper architecture.

Features include beam search decoding, multi-head attention across audio frames, and time complexity proportional to audio duration (O(n)).

5. LLM-Based Scoring (Gemini 2.5 Pro)

EchoGuard combines transcript and sound classifications into a single safety score using Gemini 2.5 Pro.

Characteristics:

- A question with a structured rubric
- Deterministic, rubric-bounded output

Architecture & Patterns

EchoGuard breaks down backend functionality into separate blueprints

- /process/vad – segmentation and preprocessing
- /process/validate-non-speech – heuristic filtering
- /cnn-model – CNN inference
- /process/transcribe – Transcription pipeline
- /process/gemini – LLM evaluation

This adheres to the Single Responsibility Principle, guaranteeing that every module contains a single, clearly defined pipeline stage.

State Management via Session Manager

- Only audio segments, confirmed non-speech outcomes, transcription, and aggregated model outputs are stored via a lightweight session manager.
- This architecture preserves per-recording context while enabling stateless endpoint calls.

Concurrent Processing in Real Time

- EchoGuard provides low-latency user interface feedback and live CNN classification by segmenting audio into 5-second intervals.

Engineering Evidence

Performance Considerations

Development-related benchmarks reveal that:

- RNNoise inference per chunk finishes in less than 50 ms
- CNN categorization takes about 20 to 40 CPU seconds.
- A 20-second footage can be transcribed by Whisper in less than two seconds.

Scalability

The audio-on-demand business model of EchoGuard guarantees that:

- no long-term audio storage takes place
- Chunk-based batch processing makes effective use of the CPU, and every recording session is completely separated.

Reproducibility

All pipelines (VAD, validation, CNN inference, transcription, scoring) can be reproduced locally using:

- Weights for the CNN model.
- Fixed logic for resampling and preprocessing (torchaudio).
- Kept quick templates for the prompting of Gemini.

Implementation Notes

Repository Structure

EchoGuard is a hybrid project that combines Flask with Next.js. All frontend files are located in the root directory, and the complete backend is contained in a separate api/ directory. This approach allows for unified deployment using frameworks like Next.js and Vercel while maintaining a clear separation between the web application and backend server.

The repository is structured like this:

```
/api              → Flask backend (core audio pipeline)
/app              → Next.js routing and page components
/components       → Reusable frontend UI components
/lib              → Frontend utility functions
/public           → Static assets for frontend
...
vercel.json       → Deployment configuration
requirements.txt  → Python dependencies
```

Backend substructure:

Within api/, the backend is divided into three major folders:

```
/api
├── assets   → CNN model weights, feature transforms
├── routes   → Flask blueprints for all endpoints
└── utils    → Shared utilities for VAD, preprocessing, merging, etc.
```

Design Choices & Tradeoffs

1. Chunk-Based Processing (5-Second Clips)

- Real-time inference of CNN models is CPU-heavy; chunking allows overlapping tasks and incremental feedback.
- Tradeoff: Clip boundaries may occasionally split meaningful sound events.

2. RNNoise for VAD instead of other VAD models

- RNNoise is lightweight, faster, and trained on noisy audio
- Tradeoff: Can misclassify low-volume speech.

3. Faster-Whisper Instead of Full Whisper Model

- Faster-Whisper is lightweight and all we needed for the context of our project.
- Tradeoff: Slight quality tradeoff vs. large Whisper models.

4. Session-Based Temporary Storage

- Simplicity for MVP purposes.
- Tradeoff: Prevents information from being persisted through multiple sessions.

Module Layout

Backend Modules:

1. server.py – Entry point to start Flask, load models, and register routes
2. routes/vad_primary.py – Handles VAD segmentation and preprocessing
3. routes/validate_non_speech.py – Heuristic validation logic
4. routes/model_router.py – CNN model inference endpoint
5. routes/transcription.py – Merges raw audio and runs Faster-Whisper
6. routes/gemini_analysis.py – Structured prompt assembly and Gemini evaluation

Frontend Modules:

1. app/ – Main pages (/record, /upload, /results)
2. components/ – UI building blocks (analysis indicators, loaders, banners)
3. lib/ – Audio recording, chunking, and POST requests
4. public/ – Static icons, images, configuration

Testing & Evaluation

The goal of EchoGuard's testing and evaluation procedure was to make sure that every phase of the audio-processing pipeline operated as planned and that the system generated consistent outcomes for all input types. The majority of testing was done using a combination of thorough manual review and lightweight unit tests.

Unit Testing

Waveform normalization, VAD segmentation logic, and non-speech validation criteria are just a few of the focused unit tests that were used to verify the performance of key utility functions. When changes were made to validation parameters or model integration, these tests helped guarantee the accuracy of individual components and avoided regressions.

Manual Testing

Throughout development, most system validation was done manually. This included:

- Confirming that during live recording, environmental sound predictions were accurate
- Evaluating transcribed results
- Assessing the coherence and consistency of risk/benefit summaries

Given the system's real-time functionality and the unpredictability of user-recorded audio, manual testing was particularly crucial.

Descriptive Logging

During development, a lot of descriptive print-style logging was employed to track the pipeline's data flow. Logging provided insights into:

- VAD decisions on each 5-second clip

- validation outcomes (pass/fail and threshold checks)
- CNN classification probabilities
- Transcription progress
- The final JSON payload used for Gemini 2.5 Pro evaluation

By using this method, the team was able to promptly detect problems, adjust validation algorithms, and guarantee that every module generated the desired results.

Security, Privacy, Accessibility & Compliance

The impact of children's media consumption on society was taken into consideration when designing EchoGuard. The main goal of the technology is to assist parents in making better decisions regarding content moderation and safety by giving them a greater understanding of the audio environments to which their kids are exposed. EchoGuard helps families navigate an increasingly media-rich digital world by offering clear, real-time insights about media that their kids are consuming.

Security Considerations & Accessibility

Despite not being a security-critical program, EchoGuard was developed using fundamental privacy-focused practices:

- No long-term storage of raw audio is done; instead, audio is processed on demand and only momentarily stored during the session.
- All temporary audio files and interim results are promptly erased following analysis.
- After database integration is added, only derived results (transcript, model outputs) might be retained in subsequent iterations.

These design decisions lessen the possibility of misuse or illegal access while still protecting user data.

Additionally, because of the Next.js interface's deliberate simplicity and minimalism, users with different levels of technical expertise can easily traverse the system. A wide range of users, including non-technical caregivers, can utilize EchoGuard since the interface offers clear feedback during recording, processing, and result creation.

Deployment & Operations

Deployment Overview

Vercel is used to deploy EchoGuard. Its serverless API routing model serves both the Flask backend and the Next.js frontend. The backend is accessible via endpoints found under the project's api/ directory, while the frontend is provided as a static, server-rendered Next.js application. The system may be deployed with little configuration because to Vercel's automatic handling of build processes, environment variable management, and routing.

Vercel's automatic build and deployment process is triggered by pushing modifications to the main GitHub branch, and the revised frontend and backend endpoints are provisioned globally.

This configuration eliminates the need for manual server management and enables rapid and reliable system updates.

Runtime Behavior

After deployment, the system functions mostly without human supervision. Vercel manages the execution of backend endpoints as serverless functions, scales API requests automatically, and provides frontend pages.

Examples of operational behavior include:

- The streaming of audio recordings in 5-second increments to the backend.

- Serverless functions that execute CNN inference, transcription, VAD, validation, and Gemini scoring on demand and provide real-time results to the frontend without the need for permanent background processes
- At the conclusion of each session, clearing temporary data

There is no long-running server and no need for maintenance duties like database administration or container management because audio is handled completely within serverless execution environments.

Limitations & Future Work

Limitations

1. Lack of Persistent Storage

- Database integration is not yet included in the system. Consequently, risk assessments, transcripts, and user results are not retained between sessions. After processing is finished, all data is erased.

2. Low-Quality Browser Audio Capture

- WebM capture, which is powered by FFmpeg and significantly decreases audio clarity, is the basis for browser-based audio recording. This has an impact on the accuracy of transcription and environmental sound classification, especially for quiet or mixed audio sources.

3. Limited CNN Sound Classes

- A limited number of sound categories were used to train the environmental sound classifier. Although the model works well for demonstration, it is unable to accurately identify a greater range of sounds or subtle contextual cues.

4. No Source Separation Step

- Detection may be hindered by background speech or other noises in real-world situations, such as recording a TV in a busy living room. Due to its complexity and computational expense, full source separation, isolating the intended audio source while eliminating others, was not put into practice. Deep learning-based separation methods (like Demucs) and spectrum masking were outside the purview of this project.

Future Work

Future versions of the system might incorporate the following to improve and expand EchoGuard:

- Integration of the Supabase database for user accounts and permanent session history
- Enhanced datasets and more sound categories for CNN training
- Target audio streams are isolated through the use of a source-separation module.
- Using locally uploaded audio as a backup or using higher-fidelity recording techniques
- Better segmentation with enhanced VAD and validation heuristics
- Improved rubric system to account for conversational context or emotional tone

References

1. Google Research. AudioSet: An Ontology and Human-Labeled Dataset for Audio Events. Available: https://research.google.com/audioset/
2. Faster-Whisper. Efficient Whisper Inference for CPUs and GPUs. Available: https://github.com/guillaumekln/faster-whisper
3. Google DeepMind. Gemini 2.5 Pro Model Capabilities. Google GenAI Documentation, 2024. Available: https://ai.google.dev/
4. Pascual, S., et al. "Learning Sound Event Classification via Convolutional Neural Networks." ArXiv:1703.03664, 2017. Available: https://ieeexplore.ieee.org/abstract/document/9115233

Appendicies

A. Installation Guide

Prerequisites:

Before installation, ensure the following tools are installed:

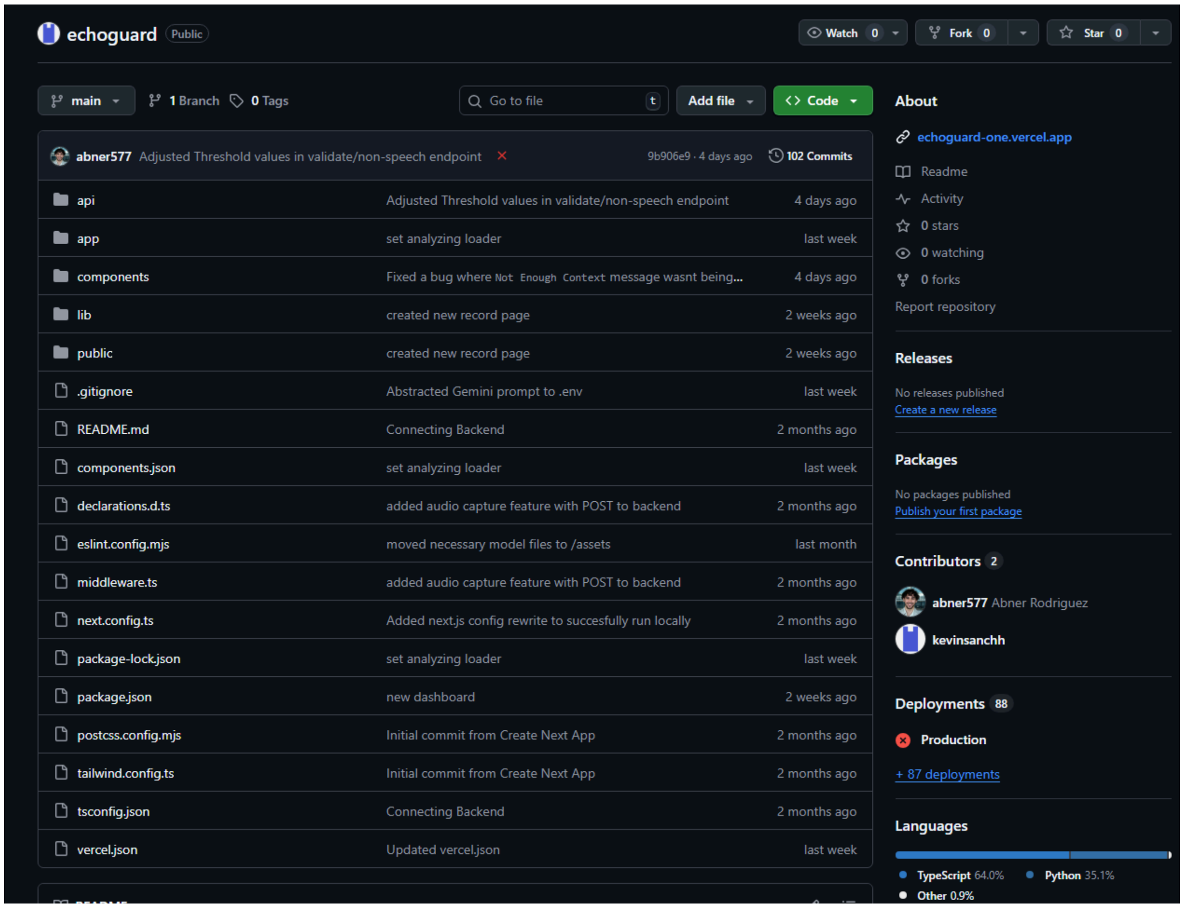
- Git
- Python 3.10+
- Node.js 18+
- npm or yarn
- Virtual environment tool, such as venv or conda

Clone the Repository

1. Open a terminal
2. Navigate to the directory where you want to install EchoGuard.
3. Clone the project repository:

- git clone https://github.com/kevinsanchh/echoguard

4. Enter the project folder → cd echoguard



abner577 Adjusted Threshold values in validate/non-speech endpoint 9b906e9 · 4 days ago 102 Commits

api	Adjusted Threshold values in validate/non-speech endpoint	4 days ago
app	set analyzing loader	last week
components	Fixed a bug where Not Enough Context message wasnt being...	4 days ago
lib	created new record page	2 weeks ago
public	created new record page	2 weeks ago
.gitignore	Abstracted Gemini prompt to .env	last week
README.md	Connecting Backend	2 months ago
components.json	set analyzing loader	last week
declarations.d.ts	added audio capture feature with POST to backend	2 months ago
eslint.config.mjs	moved necessary model files to /assets	last month
middleware.ts	added audio capture feature with POST to backend	2 months ago
next.config.ts	Added next.js config rewrite to succesfully run locally	2 months ago
package-lock.json	set analyzing loader	last week
package.json	new dashboard	2 weeks ago
postcss.config.mjs	Initial commit from Create Next App	2 months ago
tailwind.config.ts	Initial commit from Create Next App	2 months ago
tsconfig.json	Connecting Backend	2 months ago
vercel.json	Updated vercel.json	last week

About
echoguard-one.vercel.app
Readme
Activity
0 stars
0 watching
0 forks
Report repository

Releases
No releases published
Create a new release

Packages
No packages published
Publish your first package

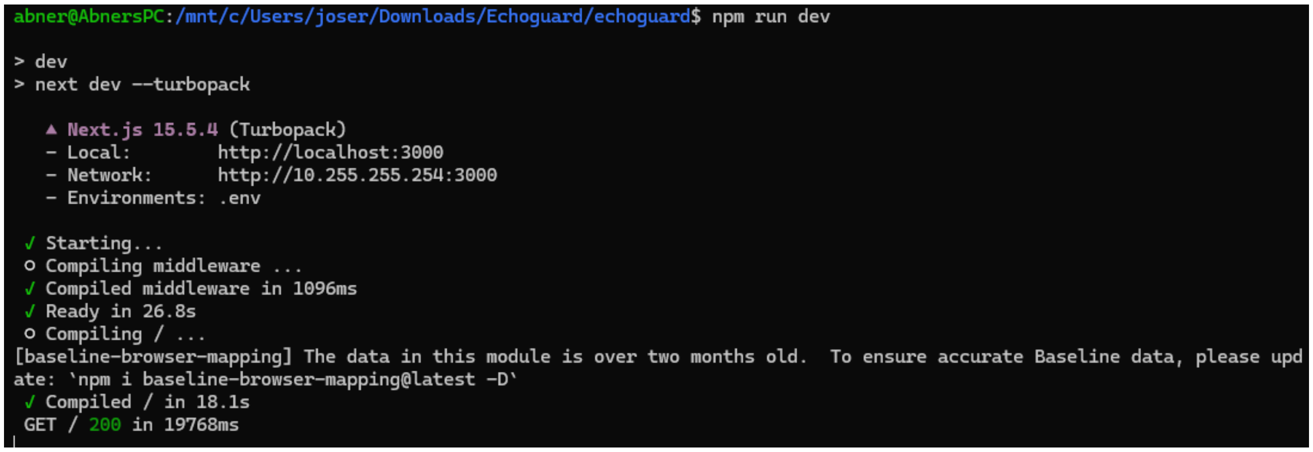
Contributors 2
abner577 Abner Rodriguez
kevinsanchh

Deployments 88
Production
+ 87 deployments

Languages
● TypeScript 64.0% ● Python 35.1%
● Other 0.9%

Install Frontend Dependencies

1. Navigate to the frontend root directory (project root) and run: npm install
2. After installation, run the frontend with: npm run dev (This starts the Next.js frontend at http://localhost:3000)



```
abner@AbnersPC:/mnt/c/Users/joser/Downloads/Echoguard/echoguard$ npm run dev

> dev
> next dev --turbopack

   ▲ Next.js 15.5.4 (Turbopack)
   - Local:        http://localhost:3000
   - Network:      http://10.255.255.254:3000
   - Environments: .env

 ✓ Starting...
 ○ Compiling middleware ...
 ✓ Compiled middleware in 1096ms
 ✓ Ready in 26.8s
 ○ Compiling / ...
[baseline-browser-mapping] The data in this module is over two months old.  To ensure accurate Baseline data, please upd
ate: `npm i baseline-browser-mapping@latest -D`
 ✓ Compiled / in 18.1s
 GET / 200 in 19768ms
```

Install Backend Dependencies

1. Navigate into the backend directory with cd api

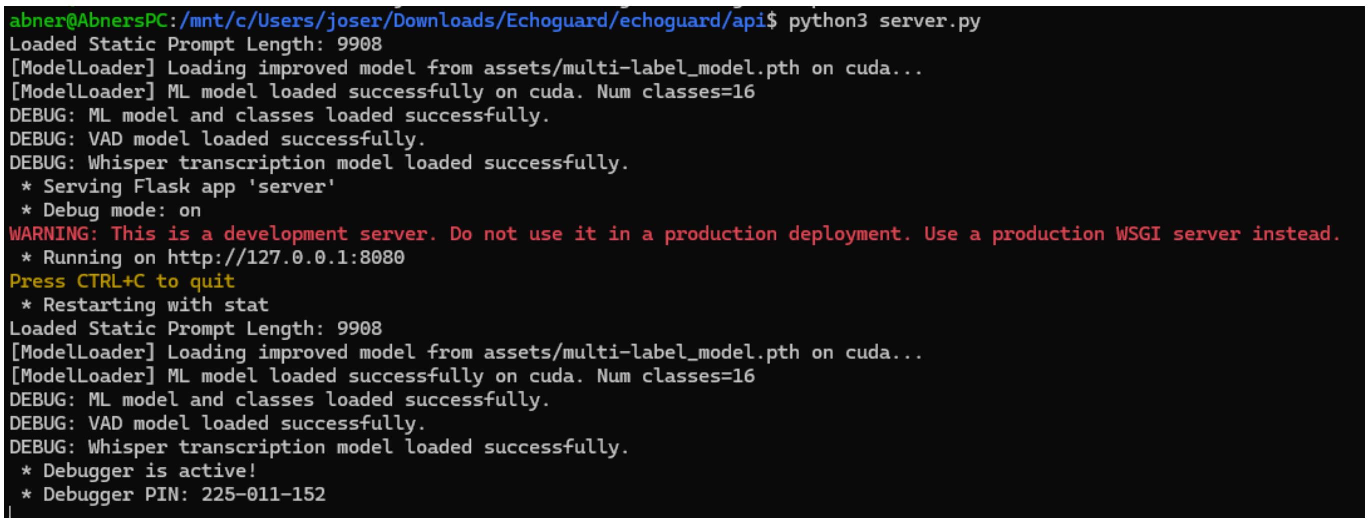
2. Create a Python virtual environment:

```
python3 -m venv venv
source venv/bin/activate   # macOS/Linux
venv\Scripts\activate      # Windows
```

3. Install Python dependencies:

- pip install -r requirements.txt

4. Start the backend server with: python3 server.py



```
abner@AbnersPC:/mnt/c/Users/joser/Downloads/Echoguard/echoguard/api$ python3 server.py
Loaded Static Prompt Length: 9908
[ModelLoader] Loading improved model from assets/multi-label_model.pth on cuda...
[ModelLoader] ML model loaded successfully on cuda. Num classes=16
DEBUG: ML model and classes loaded successfully.
DEBUG: VAD model loaded successfully.
DEBUG: Whisper transcription model loaded successfully.
 * Serving Flask app 'server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8080
Press CTRL+C to quit
 * Restarting with stat
Loaded Static Prompt Length: 9908
[ModelLoader] Loading improved model from assets/multi-label_model.pth on cuda...
[ModelLoader] ML model loaded successfully on cuda. Num classes=16
DEBUG: ML model and classes loaded successfully.
DEBUG: VAD model loaded successfully.
DEBUG: Whisper transcription model loaded successfully.
 * Debugger is active!
 * Debugger PIN: 225-011-152
```

Configure Environment Variables

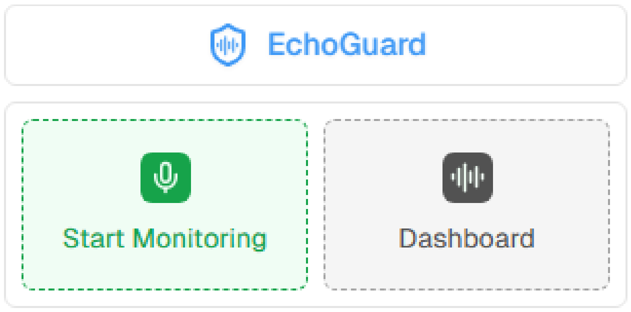
The following environment variable is required: GEMINI_API_KEY=<your-google-genai-key>

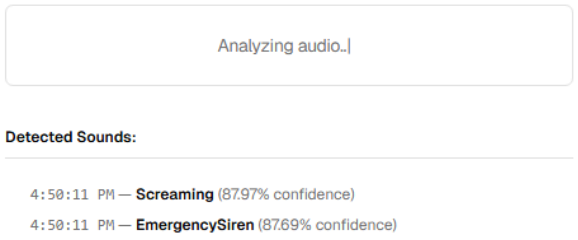
Create a .env file at the project root or configure environment variables manually.

B.User Manual

Recording Audio

1. From the home page, click “Start Recording”.
2. The browser will request microphone access — click Allow.
3. EchoGuard will begin capturing audio in 5-second chunks.
4. During recording, you will see real-time environmental sound detections appear on screen (e.g., “Crying,” “Bang,” “Emergency Siren”).
5. Click “Stop Recording” when finished.




Analyzing audio..|

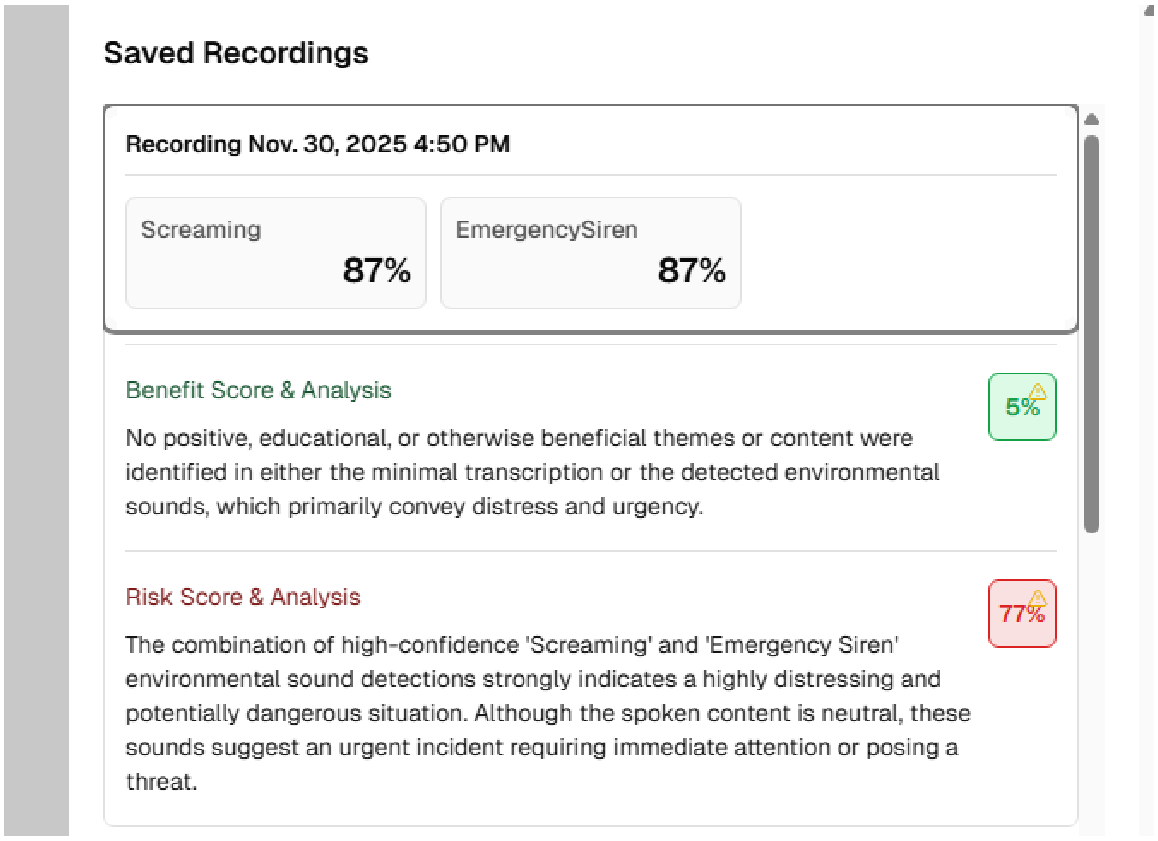
Detected Sounds:

4:50:11 PM — **Screaming** (87.97% confidence)
4:50:11 PM — **EmergencySiren** (87.69% confidence)

Viewing the Final Analysis Report (Dashboard)

After the full recording is processed, EchoGuard displays a dashboard with the following metrics:

1. Detected Environmental Sounds
2. Risk Score & Benefit Score (with Explanations)
3. Confidence Score



Saved Recordings

Recording Nov. 30, 2025 4:50 PM

Screaming 87%

EmergencySiren 87%

Benefit Score & Analysis 5%

No positive, educational, or otherwise beneficial themes or content were identified in either the minimal transcription or the detected environmental sounds, which primarily convey distress and urgency.

Risk Score & Analysis 77%

The combination of high-confidence 'Screaming' and 'Emergency Siren' environmental sound detections strongly indicates a highly distressing and potentially dangerous situation. Although the spoken content is neutral, these sounds suggest an urgent incident requiring immediate attention or posing a threat.

C. Admin/Operations Guide

Because of its lightweight backend architecture and serverless deployment strategy, EchoGuard requires relatively little operational monitoring. Verifying that the system builds correctly on Vercel and making sure environment variables are specified correctly comprise the majority of administrative work.

Deployment Monitoring

- All deployments are handled automatically by Vercel when changes are pushed to the main branch.
- Admins only need to check that the build completes successfully.

Backend Maintenance

- The backend runs as serverless functions, so no long-running servers or manual restarts are needed.
- Occasional updates to Python dependencies may require reinstalling the virtual environment locally.

Logs & Debugging

• Runtime logs and errors can be viewed directly in the Vercel dashboard under "Function Logs."

• Local debugging can be performed by running npm run dev and python3 server.py.

No On-Call or Operations Load

Because EchoGuard does not run persistent servers or background jobs, there are:

- no on-call schedules
- no uptime monitoring
- no infrastructure alerts

- no database backups

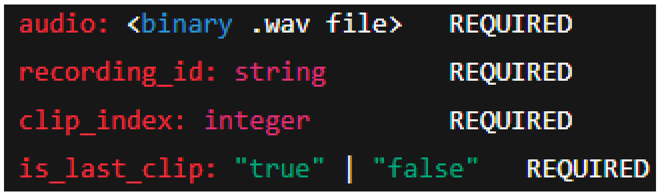
D. API Reference

Endpoint Summary

Endpoint	**Method**	**Purpose**
/process/vad	POST	Accept incoming 5-second clips, run VAD, segment audio, route non-speech for validation → CNN
/process/validate-non-speech	POST	Validate stitched non-speech audio before classification
/process/model	POST	Run CNN environmental-sound classification
/process/transcribe	POST	Merge all full clips and run Faster-Whisper transcription
/process/gemini	POST	Run Gemini 2.5 Flash safety scoring when enough context exists
/process/gemini_result	GET	Retrieve stored Gemini results (or readiness status)

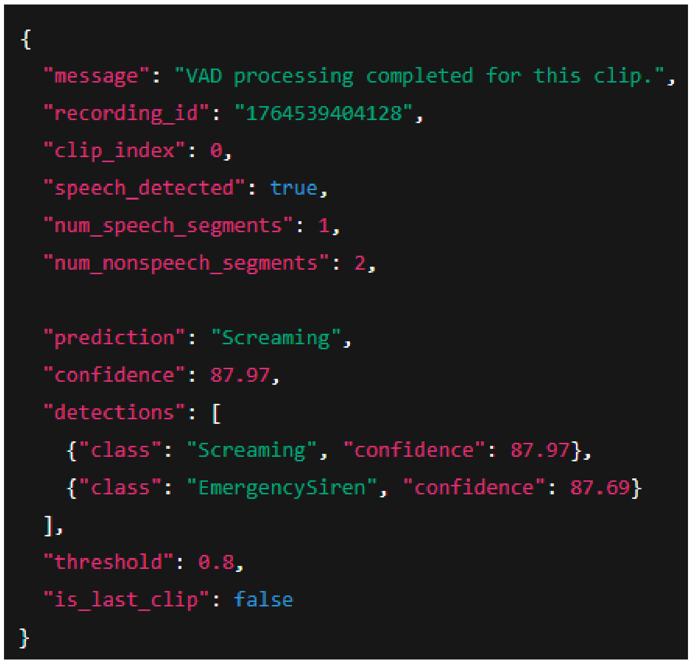
1. **/process/vad**

Request (Form-Data):



```
audio: <binary .wav file>    REQUIRED
recording_id: string         REQUIRED
clip_index: integer          REQUIRED
is_last_clip: "true" | "false"    REQUIRED
```

Response (JSON):

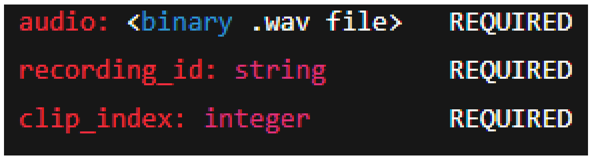


```
{
  "message": "VAD processing completed for this clip.",
  "recording_id": "1764539404128",
  "clip_index": 0,
  "speech_detected": true,
  "num_speech_segments": 1,
  "num_nonspeech_segments": 2,

  "prediction": "Screaming",
  "confidence": 87.97,
  "detections": [
    {"class": "Screaming", "confidence": 87.97},
    {"class": "EmergencySiren", "confidence": 87.69}
  ],
  "threshold": 0.8,
  "is_last_clip": false
}
```

2. /process/validate-non-speech

Request (Form-Data):



```
audio: <binary .wav file>    REQUIRED
recording_id: string         REQUIRED
clip_index: integer          REQUIRED
```

Response (JSON):

  "valid": true
}
If valid=false, the router ignores this clip and does NOT call the CNN.

3. /process/model

Request (Form-Data):

```
audio: <binary .wav file>    REQUIRED
recording_id: string         REQUIRED
clip_index: integer          REQUIRED
is_last_clip: "true" | "false"    REQUIRED
```

{Response (JSON):

```
{
  "recording_id": "1764539404128",
  "clip_index": 1,
  "prediction": "EmergencySiren",
  "confidence": 99.12,
  "detections": [
    {"class": "EmergencySiren", "confidence": 99.12}
  ],
  "threshold": 0.8,
  "is_last_clip": true
}
```

4. /process/transcribe

Request (Form-Data): recording_id: string   REQUIRED

Response (JSON):

```
{
  "recording_id": "1764539404128",
  "transcript": "Hello? Yes, yes, yes.",
  "segments": [
    {"start": 0.0, "end": 3.2, "text": "Hello? Yes, yes, yes."}
  ],
  "status": "ok"
}
```

5. /process/gemini

Request (Form-Data): recording_id: string   REQUIRED

Response (not_enough_context):

```
{
  "recording_id": "1764539404128",
  "status": "not_enough_context",
  "message": "No meaningful speech or environmental audio was detected..."
}
```

Response (waiting_for_transcription):

```
{
  "recording_id": "1764539404128",
  "status": "waiting_for_transcription"
}
```

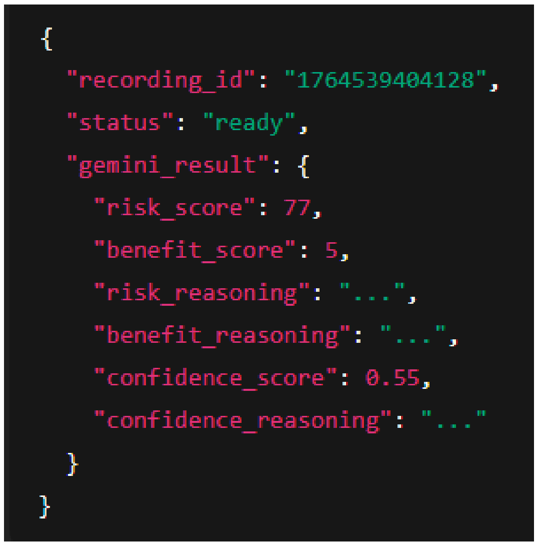
Response (ready):

```
{
  "recording_id": "1764539404128",
  "status": "completed",
  "analysis": {
    "risk_score": 77,
    "benefit_score": 5,
    "risk_reasoning": "...",
    "benefit_reasoning": "...",
    "confidence_score": 0.55,
    "confidence_reasoning": "..."
  }
}
```

6. /process/gemini_result

Query Parameters: recording_id=<id>

Response (JSON)



```
{
  "recording_id": "1764539404128",
  "status": "ready",
  "gemini_result": {
    "risk_score": 77,
    "benefit_score": 5,
    "risk_reasoning": "...",
    "benefit_reasoning": "...",
    "confidence_score": 0.55,
    "confidence_reasoning": "..."
  }
}
```

D. Poster, Slides, & Video Links

Poster & Slides:

https://drive.google.com/drive/folders/135RUxtPz08ns-WS8tne47nPCDFyjEkcs?usp=sharing

EchoGuard Introduction Video: https://youtu.be/jNRVgff8B6c?si=Z5DDLOZfh8iQD8Ym

EchoGuard Demo Video: https://youtu.be/ZLmIjZMEtM4?si=_I_Wbq_OmF-w4Unv

F. Scrum Evidence Index

https://drive.google.com/drive/folders/19L-nJh0hvwkVXXe3WYdlZ-vuwNlzC6UJ?usp=sharing