AI Interview Transcript Analyzer

AI-Driven Interview Understanding, Evaluation

& Decision Intelligence Platform

Project Overview:

The AI Interview Transcript Analyzer is an intelligent end-to-end system designed to transform raw interview audio into structured, actionable HR insights. Built to assist recruiters, HR teams, hiring managers, and talent-acquisition departments, the platform automates the entire interview-analysis workflow from transcription and diarization to sentiment scoring, skill extraction, and AI-powered candidate evaluation.

This project bridges the gap between manual interview review and data-driven decision making by converting unstructured audio into measurable, repeatable, and unbiased evaluation metrics. Powered by Whisper, Resemblyzer, advanced NLP models, Sentence Transformers, and Google Gemini, the system simulates the analysis capabilities of an experienced HR professional, providing structured summaries, skill mappings, sentiment trends, and Hire/No-Hire recommendations.

Scenario 1: HR Recruitment & Candidate Evaluation

Recruiters and hiring managers often spend hours listening to interview recordings, transcribing responses, and making subjective evaluations. The AI Interview Transcript Analyzer streamlines this process by automatically transcribing audio, separating speakers, analysing sentiment, and extracting skills directly from the candidate’s speech. For example, an HR team hiring for a customer support role can upload an interview file and instantly receive a polished transcript, emotional tone analysis, and a breakdown of the candidate’s strengths.

The system’s Gemini-powered evaluation engine generates a concise interview summary, highlights key performance areas, and provides a Hire/No-Hire recommendation with confidence scoring. This ensures assessments remain consistent and objective across all candidates. As a result, organizations can accelerate hiring cycles, reduce manual workload, and base decisions on structured insights rather than subjective impressions.

Scenario 2: Training, Coaching & Interview Preparation

Training centres, mentors, and career coaches often need to review mock interviews and provide detailed feedback to students an effort that becomes time-consuming and repetitive. With the AI Interview Transcript Analyzer, a coach can upload any recorded mock interview and instantly receive a structured transcript, sentiment breakdown, and detected skills, making evaluation faster and more accurate.

The AI-generated performance analysis helps students understand where they excelled and where improvement is needed. Section-wise evaluation, summary insights, and downloadable professional reports allow learners to track progress and prepare more effectively for real interviews. This makes the platform a powerful tool for interview practice, coaching programs, and job readiness preparation.

Architecture Overview:

The AI Interview Transcript Analyzer is built on a modular, multi-layered architecture that connects audio processing, natural language understanding, and intelligent evaluation into a seamless workflow. At its foundation, the system uses Whisper for accurate transcription and Resemblyzer for speaker diarization, enabling the platform to distinguish between interviewers and candidates with high precision. These components work together to transform raw audio into structured, timestamped segments that serve as the basis for further NLP analysis.

On top of the audio layer sits the language intelligence stack, powered by SpaCy, Sentence Transformers, and HuggingFace sentiment models. This layer performs transcript cleaning, sentiment scoring, skills extraction, entity detection, and experience parsing. These insights are then fed into the Gemini LLM, which generates the final structured evaluation including summaries, section-wise performance analysis, and hire/no-hire decisions. Each module functions independently while communicating through centralized pipeline logic, ensuring scalability and clear separation of responsibilities.

The entire system is wrapped in a sleek, interactive Streamlit interface that allows users to upload audio, run analysis, explore dashboards, and download professional reports in TXT, JSON, or PDF. Supporting utilities such as the report exporter, configuration module, and session-state handling ensure smooth UI-backend coordination. This layered architecture enables fast processing, modular extensibility, and a highly maintainable codebase ideal for both enterprise HR systems and training environments.

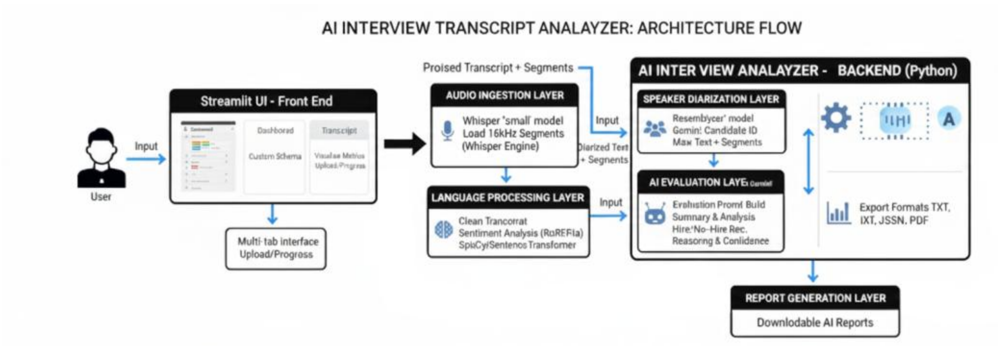


Figure 1: System architecture diagram

Core Technologies:

- **Whisper (Speech-to-Text Engine):**
  The system uses OpenAI's Whisper model to perform accurate, multilingual transcription directly from raw audio. It supports segment-level timestamps, enabling precise diarization and downstream language analysis. Whisper ensures high-quality speech recognition even in noisy environments, making the system reliable for real-world interview recordings.
- **Resemblyzer (Speaker Embedding & Diarization):**
  Resemblyzer generates voice embeddings that capture speaker identity across audio segments. These embeddings are clustered using Agglomerative Clustering to distinguish between interviewers and candidates, enabling clean separation of speech roles. This supports targeted analysis of candidate responses.
- **SpaCy & Sentence Transformers (Language Processing & Skill Extraction):**
  SpaCy performs entity extraction, phrase analysis, and text segmentation, while Sentence Transformers generate semantic embeddings for identifying skills, tools, languages, and academic qualifications. Together, they form the foundation for skill detection and structured candidate profiling.
- **HuggingFace Transformers (Sentiment Analysis):**
  A fine-tuned RoBERTa sentiment classifier evaluates emotional tone in the candidate's responses. Segment-level and overall sentiment scores provide insight into confidence, clarity, and communication style throughout the interview.
- **Google Gemini (AI Summary & Hiring Decision):**
  Gemini powers the final evaluation module, generating interview summaries, performance breakdowns, and hire/no-hire recommendations.
- **Streamlit (User Interface & Reporting):**
  Streamlit forms the front-end layer, delivering an intuitive dashboard for uploading audio, viewing transcript analytics, and downloading reports. Custom CSS styling enhances user experience, while session-state management ensures seamless multi-step analysis.
- **Python Backend & Utilities (Core Logic Layer):**
  Python modules manage audio ingestion, metadata extraction, preprocessing, diarization, skill detection, and report generation. Libraries like NumPy, Librosa, SoundFile and ReportLab support efficient audio handling and export functionality. This backend orchestrates all components into a unified processing pipeline.

Component-Wise Architecture:

Component	Description
Streamlit User Interface	Provides an interactive dashboard for uploading interview audio, running AI analysis, viewing dashboards, and downloading reports.
Audio Ingestion Module	Handles audio validation, loading, normalization, and preprocessing using Librosa. Converts audio into consistent formats suitable for Whisper transcription and diarization.
Whisper Transcription Engine	Converts audio into text with timestamped segments. Supports transcription from arrays and file paths, ensuring accurate speech-to-text conversion in various conditions.
Speaker Diarization Module	Uses Resemblyzer embeddings and clustering to identify different speakers. Determines the candidate speaker using Gemini and extracts candidate-specific segments and transcripts.
Transcript Cleaning Module	Removes fillers, normalizes punctuation, and formats segments into readable, structured text. Produces clean transcripts suitable for AI evaluation and report generation.
Sentiment Analysis Engine	Applies a RoBERTa-based sentiment classifier to evaluate emotional tone at both segment and transcript levels, generating detailed sentiment insights.
Skills & Experience Extractor	Uses SpaCy, NLTK, and Sentence Transformer embeddings to detect technical skills, tools, languages, degrees, organizations, and experience durations. Provides structured candidate profiles.
Gemini Evaluation Module	Generates interview summaries, recommendation decisions, performance breakdowns, and reasoning based on all extracted signals transcript, sentiment, skills, and diarized context.
Pipeline Orchestrator	Central controller that coordinates all modules audio ingestion, transcription, diarization, sentiment, skills extraction, and Gemini evaluation into a unified analysis flow.
Report Exporter (TXT/JSON/PDF)	Converts final results into professional reports using JSON encoding and PDF generation with ReportLab. Enables secure export of insights for HR teams and candidates.
Backend Utility & Configuration Layer	Manages API keys, model configurations (Whisper, Gemini, embeddings), skill ontologies, and global constants such as sample rate and file limits.

Pre-requisites:

1. **Python Environment Setup**
   Install Python 3.9+ and create a dedicated virtual environment for clean dependency management.

   Official Download: https://www.python.org/downloads/

2. **Install Required Libraries**
   All dependencies used in audio processing, NLP, LLM integration, and Streamlit UI must be installed from requirements.txt.

   Key libraries include: Whisper, Resemblyzer, Librosa, SpaCy, Sentence Transformers HuggingFace Transformers, Google Generative AI SDK, ReportLab, and Streamlit.

3. **Streamlit Installation**
   Required for running the interactive interview analysis dashboard.

   Docs: https://docs.streamlit.io/library/get-started/installation

4. **NLP & Embedding Models Setup**
   - SpaCy language model (en_core_web_lg)
   - Sentence Transformer model (all-MiniLM-L6-v2)
   - HuggingFace sentiment model

     SpaCy Models: https://spacy.io/models

     Sentence Transformers: https://www.sbert.net/

5. **Google Gemini API Key**
   Required for summary generation and AI decision-making.

   Google Generative AI Setup: https://ai.google.dev/

6. **Optional Tools for Development**
   Recommended IDEs:
   - Visual Studio Code: https://code.visualstudio.com/
   - PyCharm Community: https://www.jetbrains.com/pycharm/

Project Flow:

1. Environment Setup and Dependency Configuration

- **Activity 1.1:** Obtain and configure the Google Gemini API Key, set up Hugging Face access, and prepare environment variables in .env.

- **Activity 1.2:** Install all required dependencies (Whisper, Resemblyzer, Spacy, HuggingFace models, Streamlit) and validate model loading.

- **Activity 1.3:** Create the project folder structure and configure the config.py file with model names, API keys, and global settings.

2. Core Audio & Language Processing Pipeline Development

- **Activity 2.1:** Implement audio ingestion, format validation, preprocessing, and normalization using Librosa.

- **Activity 2.2:** Integrate the Whisper transcription engine to generate timestamped transcript segments.

- **Activity 2.3:** Implement speaker diarization using Resemblyzer embeddings and clustering; detect candidate speaker and extract candidate-specific segments.

- **Activity 2.4:** Build modules for transcript cleaning, sentiment analysis, skill extraction, and linguistic profiling.

3. AI-Powered HR Evaluation and Decision Logic

- **Activity 3.1:** Engineer evaluation prompts combining transcript, sentiment, skills, and speaker metadata for Gemini.

- **Activity 3.2:** Generate final interview summary, section-wise evaluation, hire/no-hire decision, and reasoning using Gemini.

- **Activity 3.3:** Aggregate all pipeline outputs (transcript, skills, sentiment, metrics) into structured results for UI and reporting.

4. Streamlit UI Implementation and User Interaction

- **Activity 4.1:** Build interactive layout with session state, custom styling, and animated components.

- **Activity 4.2:** Implement multi-tab interface (Dashboard, AI Evaluation, Skills Analysis, Transcript Viewer).

- **Activity 4.3:** Develop the audio upload system, progress indicators, and error handling for the analysis workflow.

- **Activity 4.4:** Display insights and visualizations including transcript segments, skill tags, sentiment metrics, and evaluation results.

- **Activity 4.5:** Implement the report export system to download TXT, JSON, and PDF reports.

5. Testing, Optimization, and Deployment

- **Activity 5.1:** Test UI components, verify audio upload, and validate correct display of metrics, transcript, skills, and evaluation.

- **Activity 5.2:** Conduct deployment preparation by verifying environment variables, optimizing CSS and assets, and ensuring stable end-to-end execution.

- **Activity 5.3:** Perform full pipeline testing on the deployed application, ensuring cross-browser compatibility, responsive UI, and error-free report downloads.

MILESTONE 1: Environment Setup and Dependency Configuration

This foundational milestone establishes the technical environment required for building and deploying the Synthetic Data Factory (SDF). It ensures that all dependencies, frameworks, and integrations are configured correctly for seamless execution of synthetic data generation, visualization, and validation workflows.

Activity 1.1: Obtain Google Gemini API Key

- Sign in or create an account on the official Google AI Studio platform https://aistudio.google.com/

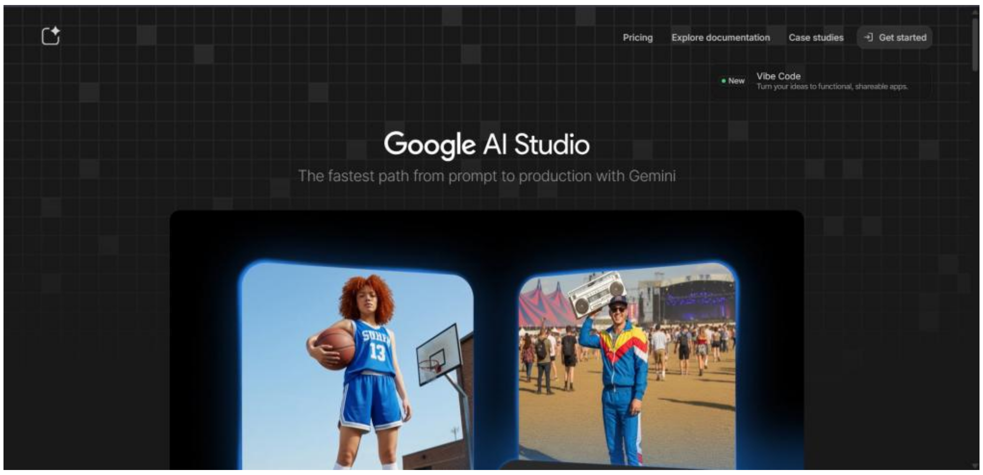

Pricing
Explore documentation
Case studies
Get started
New
Vibe Code
Turn your ideas to functional, shareable apps.
Google AI Studio
The fastest path from prompt to production with Gemini

Figure 2: Google AI Studio Console

- Clicking on "Get Started" that's leads to this Google AI Dashboard:

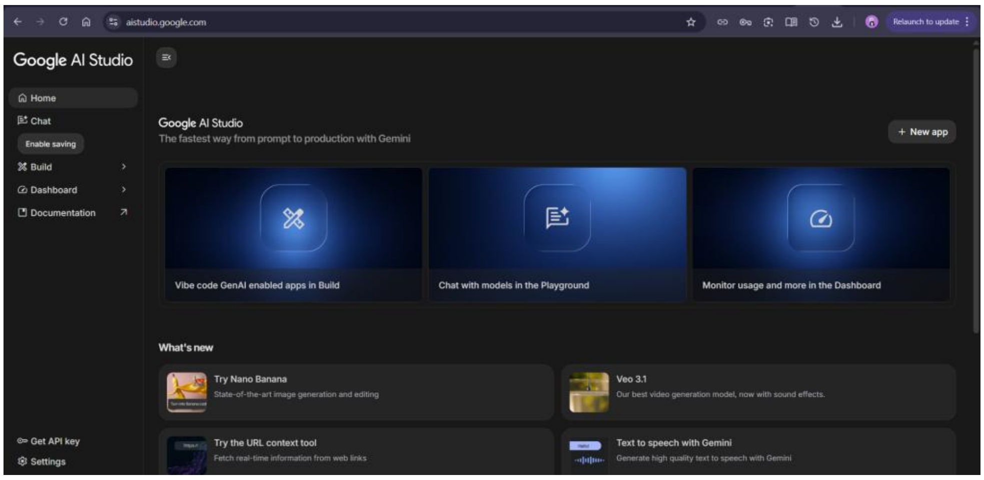
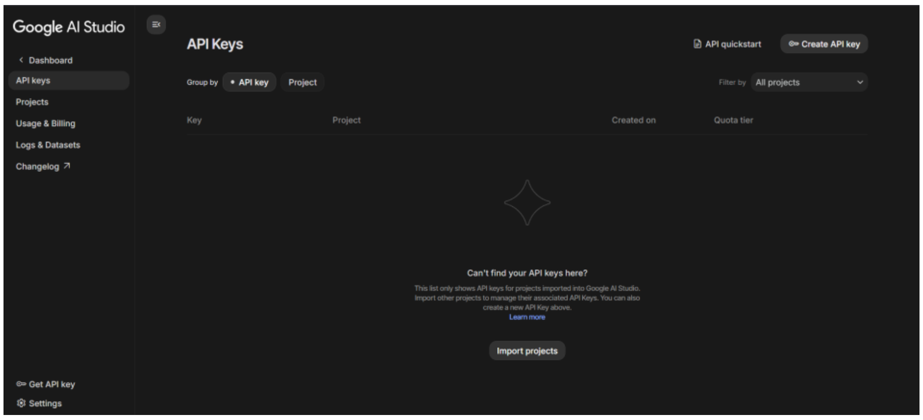

aistudio.google.com
Relaunch to update
Google AI Studio
Home
Chat
Enable saving
Build
Dashboard
Documentation
Get API key
Settings
Google AI Studio
The fastest way from prompt to production with Gemini
New app
Vibe code GenAI enabled apps in Build
Chat with models in the Playground
Monitor usage and more in the Dashboard
What's new
Try Nano Banana
State-of-the-art image generation and editing
Veo 3.1
Our best video generation model, now with sound effects.
Try the URL context tool
Fetch real-time information from web links
Text to speech with Gemini
Generate high quality text to speech with Gemini

Figure 3: Google AI Studio Dashboard

- Navigate to the Get API Keys section under your account dashboard.

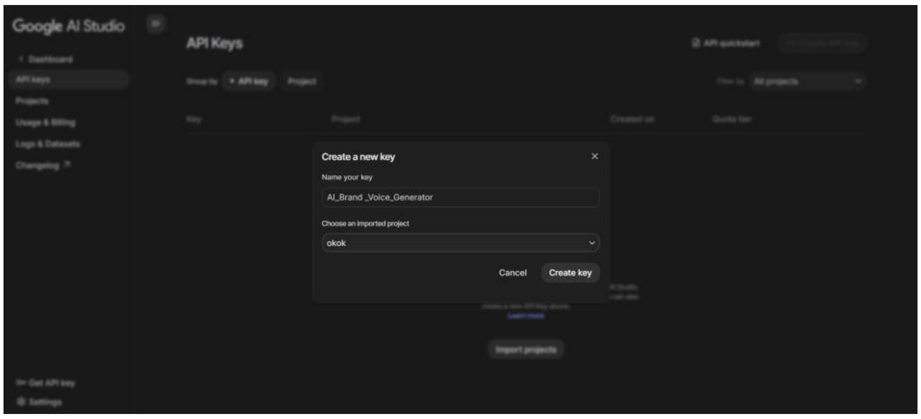


Can't find your API keys here?

This list only shows API keys for projects imported into Google AI Studio. Import other projects to manage their associated API Keys. You can also create a new API Key above.

Figure 4: API Key Management Console

- Click on **"Create API Key"** to generate a new key for your project.
- Name the key appropriately (e.g., AI_Interview_Analyser) for easy identification.

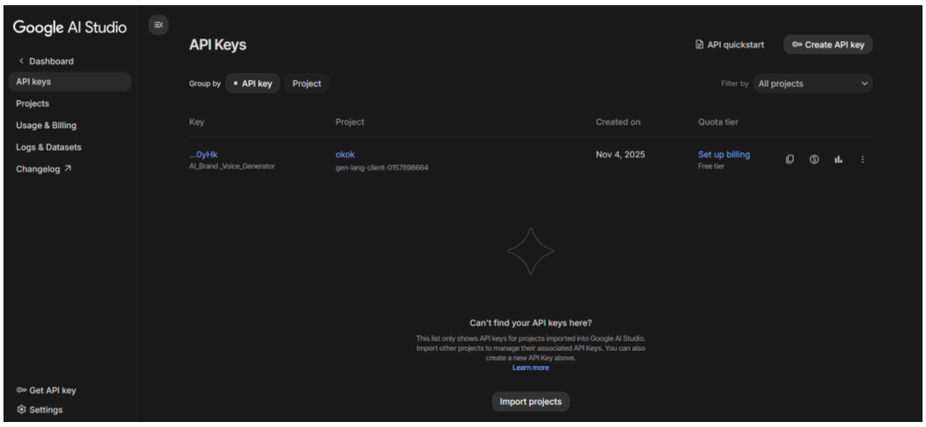


Name your key: AI_Brand _Voice_Generator

Choose an imported project: okok

Figure 5: Creating API Key

- Copy and securely store the generated API key for later configuration in your .env file.



Key	Project	Created on	Quota tier
...0yHk AI_Brand_Voice_Generator	okok gen-lang-client-0157898664	Nov 4, 2025	Set up billing Free tier

Can't find your API keys here?

This list only shows API keys for projects imported into Google AI Studio. Import other projects to manage their associated API Keys. You can also create a new API Key above.

Figure 6: Showing the API Key and Project name on Google Console

Activity 1.2: Set Up Hugging Face Token and Model Access

- Create account or sign in to Hugging Face platform https://huggingface.co/

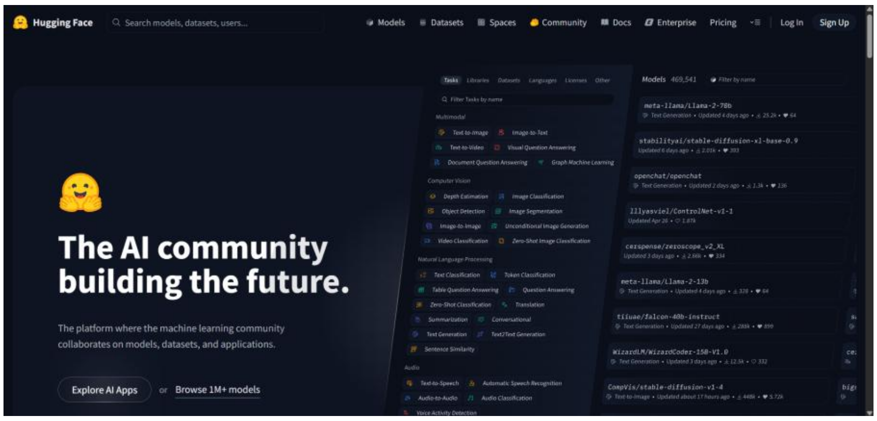
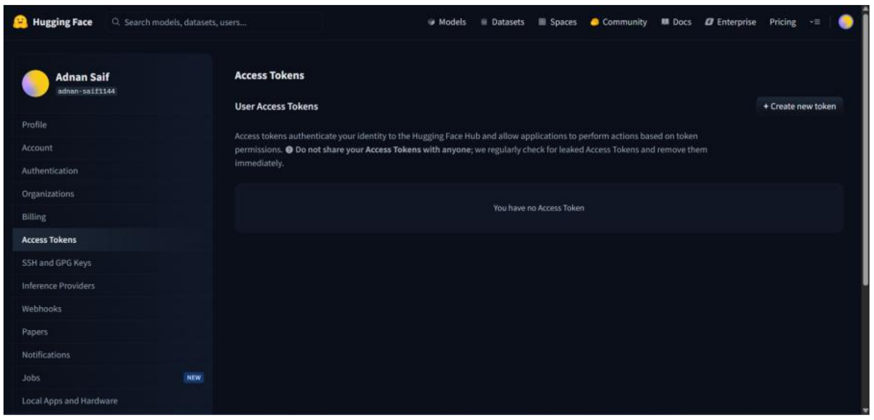

Hugging Face
Models
Datasets
Spaces
Community
Docs
Enterprise
Pricing
Log In
Sign Up
The AI community building the future.
The platform where the machine learning community collaborates on models, datasets, and applications.
Explore AI Apps
Browse 1M+ models

Figure 7: Hugging Face home page

- Navigate to Settings → Access Tokens in your account dashboard



Access Tokens

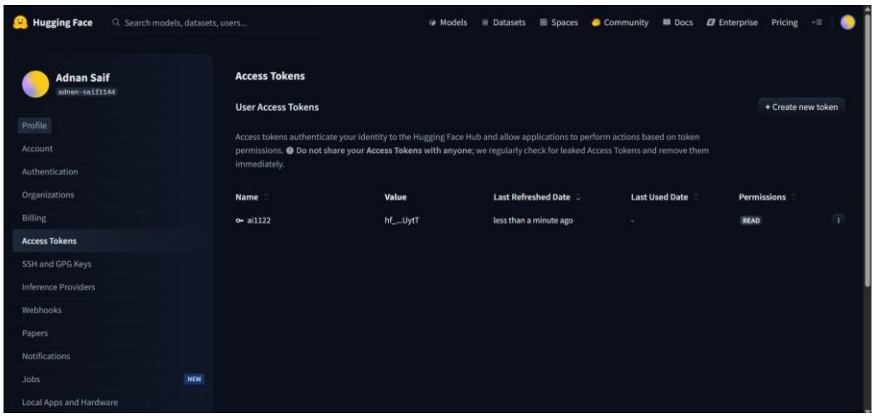
User Access Tokens

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions. **Do not share your Access Tokens with anyone**; we regularly check for leaked Access Tokens and remove them immediately.

You have no Access Token

Figure 8: Creating Access Token

- Generate a new token with appropriate permissions for model access



Access Tokens

User Access Tokens

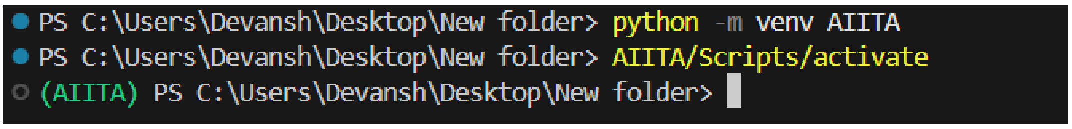
Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions. **Do not share your Access Tokens with anyone**; we regularly check for leaked Access Tokens and remove them immediately.

Name	Value	Last Refreshed Date	Last Used Date	Permissions
ai1122	hf_...UytT	less than a minute ago	-	READ

Figure 9: Hugging Face Token

Activity 1.3: Environment Setup & Dependency Installation

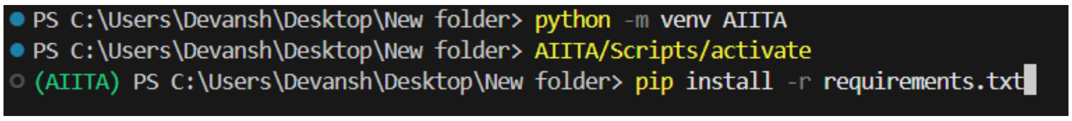
- Create a virtual environment for the project:



```
PS C:\Users\Devansh\Desktop\New folder> python -m venv AIITA
PS C:\Users\Devansh\Desktop\New folder> AIITA/Scripts/activate
(AIITA) PS C:\Users\Devansh\Desktop\New folder>
```

Figure 10: Creating & Activating Environment

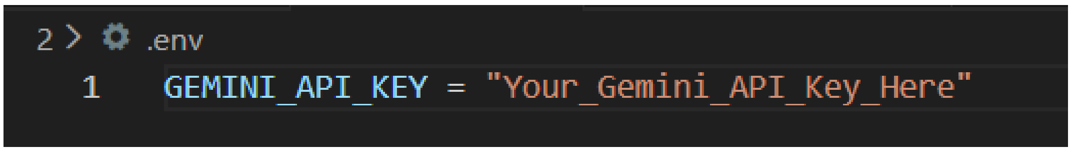
- Install project dependencies:



```
PS C:\Users\Devansh\Desktop\New folder> python -m venv AIITA
PS C:\Users\Devansh\Desktop\New folder> AIITA/Scripts/activate
(AIITA) PS C:\Users\Devansh\Desktop\New folder> pip install -r requirements.txt
```

Figure 11: Installing requirements

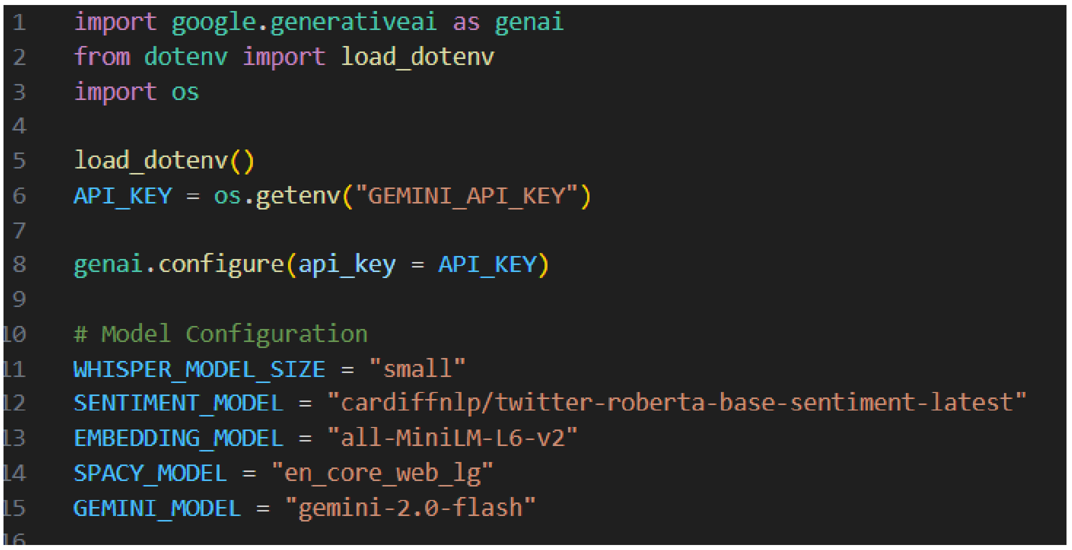
- Making a .env for securing the Gemini API Key as paste the api key here:



```
2 > .env
GEMINI_API_KEY = "Your_Gemini_API_Key_Here"
```

Figure 12: .env File

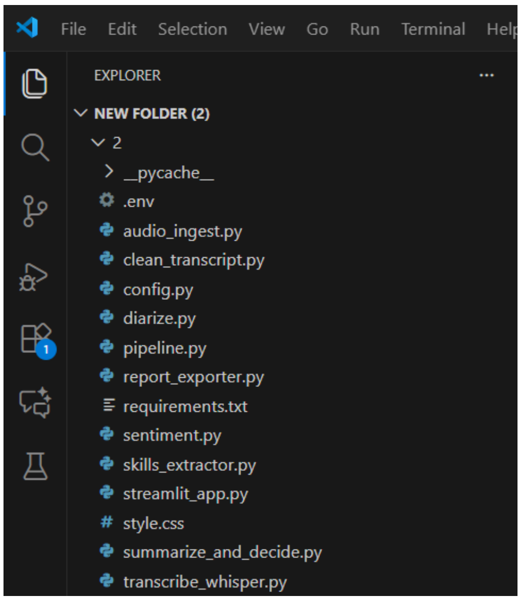
- Configure all environment variables in config.py:
  - GEMINI_API_KEY
  - Model names



```python
import google.generativeai as genai
from dotenv import load_dotenv
import os

load_dotenv()
API_KEY = os.getenv("GEMINI_API_KEY")

genai.configure(api_key = API_KEY)

# Model Configuration
WHISPER_MODEL_SIZE = "small"
SENTIMENT_MODEL = "cardiffnlp/twitter-roberta-base-sentiment-latest"
EMBEDDING_MODEL = "all-MiniLM-L6-v2"
SPACY_MODEL = "en_core_web_lg"
GEMINI_MODEL = "gemini-2.0-flash"
```

Figure 13: Configure Models

- Set up the project structure for modular development:



```
File  Edit  Selection  View  Go  Run  Terminal  Help

EXPLORER
NEW FOLDER (2)
  2
    __pycache__
    .env
    audio_ingest.py
    clean_transcript.py
    config.py
    diarize.py
    pipeline.py
    report_exporter.py
    requirements.txt
    sentiment.py
    skills_extractor.py
    streamlit_app.py
    style.css
    summarize_and_decide.py
    transcribe_whisper.py
```

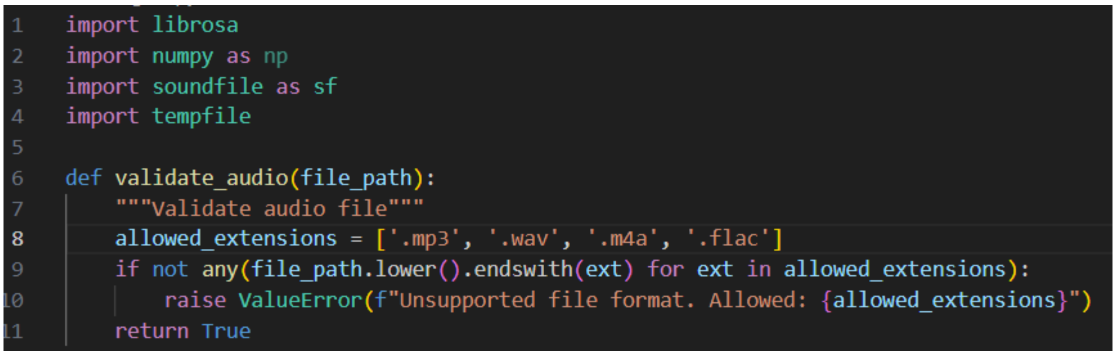
Figure 14: Folder Structure

MILESTONE 2: Core Audio & Language Processing Pipeline Development

This milestone builds the backbone of the analysis system extracting speech, identifying speakers, cleaning transcripts, measuring sentiment, and parsing linguistic details. These components work together to transform raw audio into structured, analysable data.

Activity 2.1: Audio Ingestion & Preprocessing

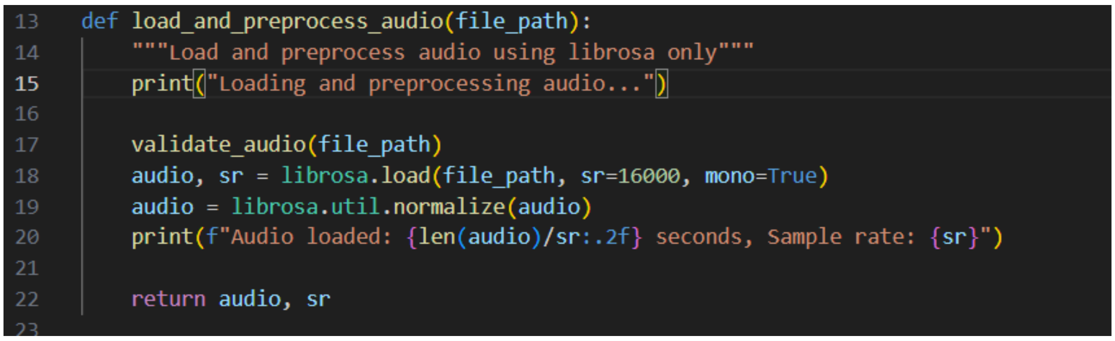
- Implement audio format validation (.mp3/.wav/.m4a).



```python
import librosa
import numpy as np
import soundfile as sf
import tempfile

def validate_audio(file_path):
    """Validate audio file"""
    allowed_extensions = ['.mp3', '.wav', '.m4a', '.flac']
    if not any(file_path.lower().endswith(ext) for ext in allowed_extensions):
        raise ValueError(f"Unsupported file format. Allowed: {allowed_extensions}")
    return True
```

Figure 15: Validating audio file

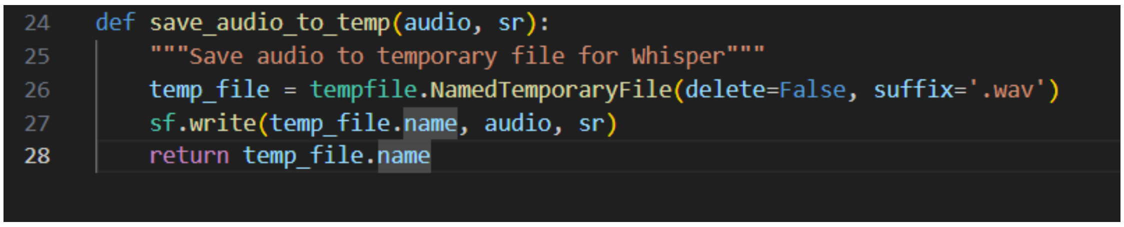
- Load audio with Librosa at 16kHz and normalize amplitude.



```python
def load_and_preprocess_audio(file_path):
    """Load and preprocess audio using librosa only"""
    print("Loading and preprocessing audio...")

    validate_audio(file_path)
    audio, sr = librosa.load(file_path, sr=16000, mono=True)
    audio = librosa.util.normalize(audio)
    print(f"Audio loaded: {len(audio)/sr:.2f} seconds, Sample rate: {sr}")

    return audio, sr
```

Figure 16: Normalize audio

- Save processed audio into temporary WAV files for Whisper.

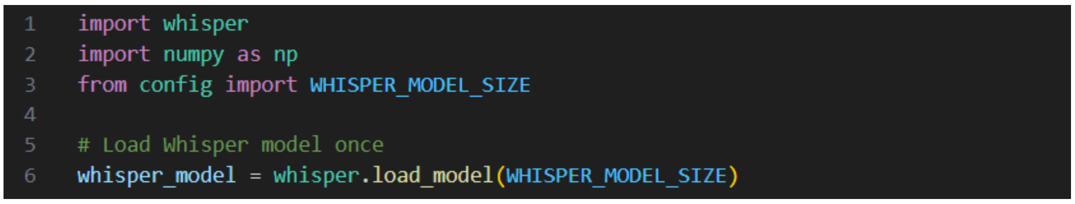


```python
def save_audio_to_temp(audio, sr):
    """Save audio to temporary file for Whisper"""
    temp_file = tempfile.NamedTemporaryFile(delete=False, suffix='.wav')
    sf.write(temp_file.name, audio, sr)
    return temp_file.name
```

Figure 17: Saving processed audio

Activity 2.2: Whisper Transcription Engine Implementation

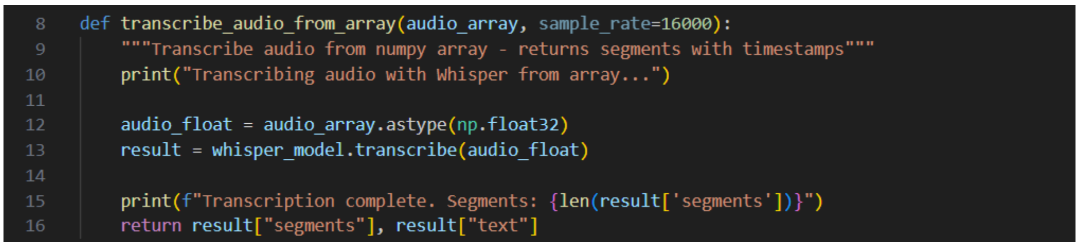
- Integrate Whisper “small” model for fast transcription.



```python
import whisper
import numpy as np
from config import WHISPER_MODEL_SIZE

# Load Whisper model once
whisper_model = whisper.load_model(WHISPER_MODEL_SIZE)
```

Figure 18: Loading Whisper model

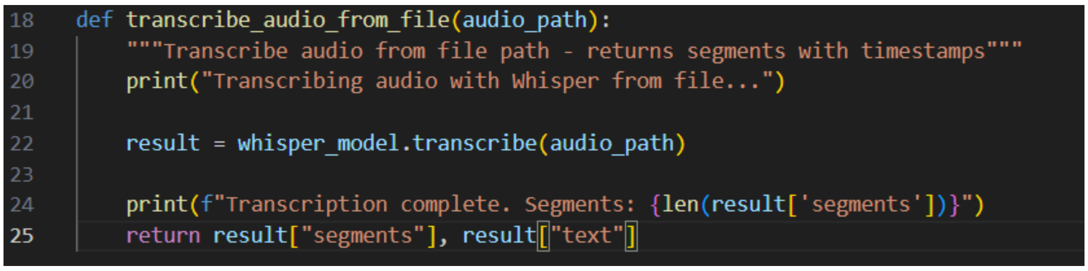
- Generate transcript segments with timestamps.



```python
def transcribe_audio_from_array(audio_array, sample_rate=16000):
    """Transcribe audio from numpy array - returns segments with timestamps"""
    print("Transcribing audio with Whisper from array...")

    audio_float = audio_array.astype(np.float32)
    result = whisper_model.transcribe(audio_float)

    print(f"Transcription complete. Segments: {len(result['segments'])}")
    return result["segments"], result["text"]
```

Figure19: Generating Transcript with array

- Validate transcription accuracy on sample audio.

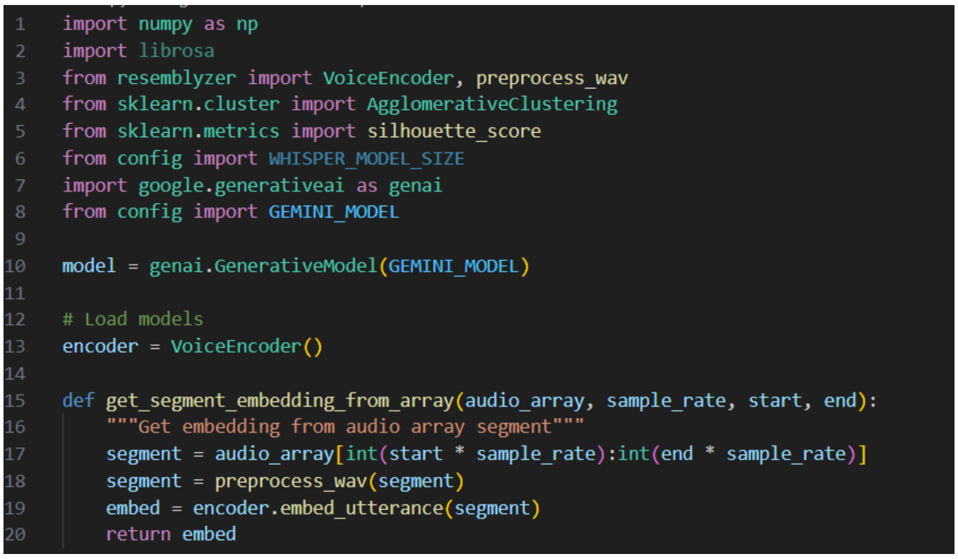


```python
def transcribe_audio_from_file(audio_path):
    """Transcribe audio from file path - returns segments with timestamps"""
    print("Transcribing audio with Whisper from file...")

    result = whisper_model.transcribe(audio_path)

    print(f"Transcription complete. Segments: {len(result['segments'])}")
    return result["segments"], result["text"]
```

Figure 20: Transcript with audio

Activity 2.3: Speaker Diarization System

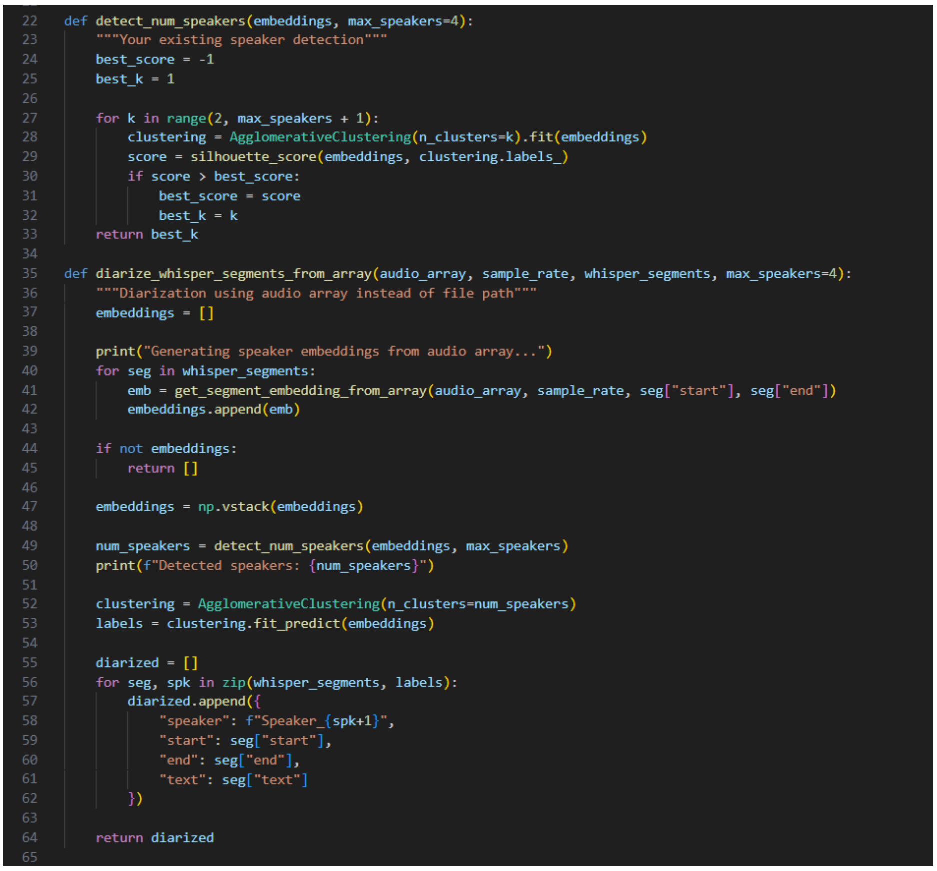
- Extract speaker embeddings using Resemblyzer.



```python
import numpy as np
import librosa
from resemblyzer import VoiceEncoder, preprocess_wav
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from config import WHISPER_MODEL_SIZE
import google.generativeai as genai
from config import GEMINI_MODEL

model = genai.GenerativeModel(GEMINI_MODEL)

# Load models
encoder = VoiceEncoder()

def get_segment_embedding_from_array(audio_array, sample_rate, start, end):
    """Get embedding from audio array segment"""
    segment = audio_array[int(start * sample_rate):int(end * sample_rate)]
    segment = preprocess_wav(segment)
    embed = encoder.embed_utterance(segment)
    return embed
```

Figure 21: Embedding with resemblyzer

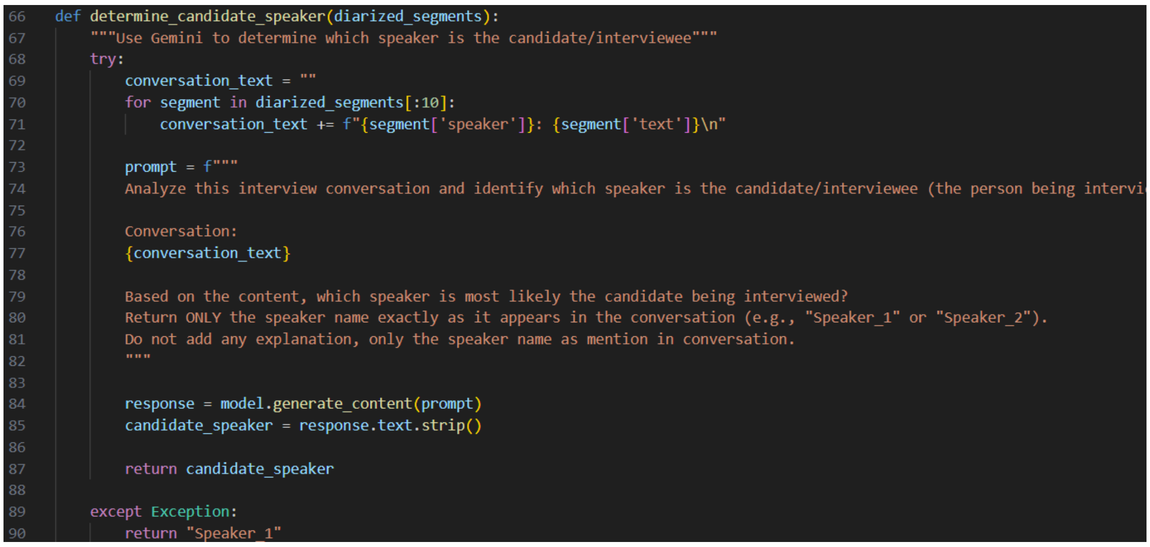
- Perform clustering using Agglomerative Clustering.



```python
def detect_num_speakers(embeddings, max_speakers=4):
    """Your existing speaker detection"""
    best_score = -1
    best_k = 1

    for k in range(2, max_speakers + 1):
        clustering = AgglomerativeClustering(n_clusters=k).fit(embeddings)
        score = silhouette_score(embeddings, clustering.labels_)
        if score > best_score:
            best_score = score
            best_k = k
    return best_k

def diarize_whisper_segments_from_array(audio_array, sample_rate, whisper_segments, max_speakers=4):
    """Diarization using audio array instead of file path"""
    embeddings = []

    print("Generating speaker embeddings from audio array...")
    for seg in whisper_segments:
        emb = get_segment_embedding_from_array(audio_array, sample_rate, seg["start"], seg["end"])
        embeddings.append(emb)

    if not embeddings:
        return []

    embeddings = np.vstack(embeddings)

    num_speakers = detect_num_speakers(embeddings, max_speakers)
    print(f"Detected speakers: {num_speakers}")

    clustering = AgglomerativeClustering(n_clusters=num_speakers)
    labels = clustering.fit_predict(embeddings)

    diarized = []
    for seg, spk in zip(whisper_segments, labels):
        diarized.append({
            "speaker": f"Speaker_{spk+1}",
            "start": seg["start"],
            "end": seg["end"],
            "text": seg["text"]
        })

    return diarized
```

Figure 22: Clustering the speaker

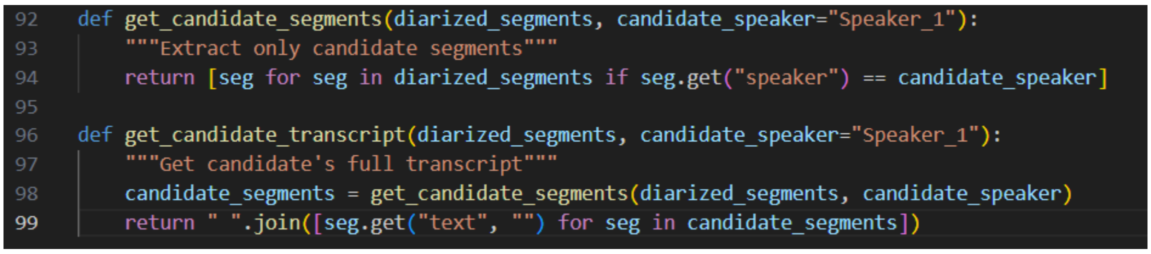
- Auto-detect candidate speaker using Gemini by analysing early conversation segments.



```python
def determine_candidate_speaker(diarized_segments):
    """Use Gemini to determine which speaker is the candidate/interviewee"""
    try:
        conversation_text = ""
        for segment in diarized_segments[:10]:
            conversation_text += f"{segment['speaker']}: {segment['text']}\n"

        prompt = f"""
        Analyze this interview conversation and identify which speaker is the candidate/interviewee (the person being intervi

        Conversation:
        {conversation_text}

        Based on the content, which speaker is most likely the candidate being interviewed?
        Return ONLY the speaker name exactly as it appears in the conversation (e.g., "Speaker_1" or "Speaker_2").
        Do not add any explanation, only the speaker name as mention in conversation.
        """

        response = model.generate_content(prompt)
        candidate_speaker = response.text.strip()

        return candidate_speaker

    except Exception:
        return "Speaker_1"
```

Figure 23: Detection of Candidate

- Candidate diarized segments with Whisper timestamps.

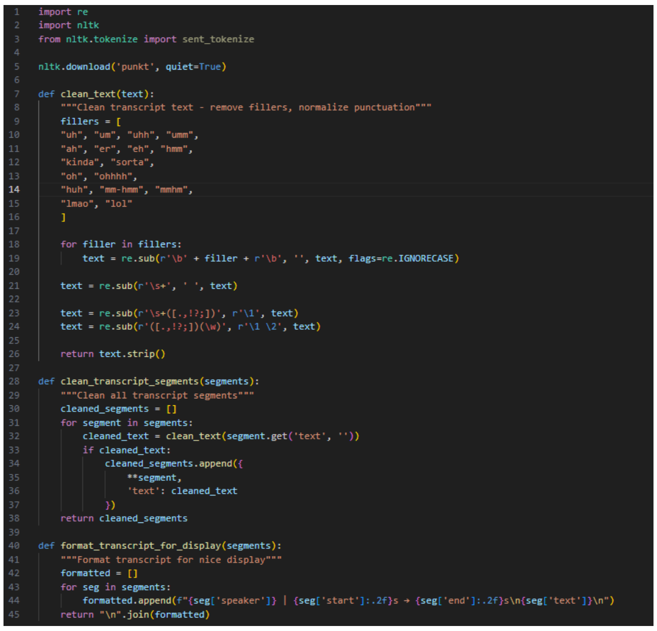


```python
def get_candidate_segments(diarized_segments, candidate_speaker="Speaker_1"):
    """Extract only candidate segments"""
    return [seg for seg in diarized_segments if seg.get("speaker") == candidate_speaker]

def get_candidate_transcript(diarized_segments, candidate_speaker="Speaker_1"):
    """Get candidate's full transcript"""
    candidate_segments = get_candidate_segments(diarized_segments, candidate_speaker)
    return " ".join([seg.get("text", "") for seg in candidate_segments])
```

Figure 24: Extracting Candidate Segments

Activity 2.4: Transcript Cleaning, Sentiment Processing & Skill Extraction

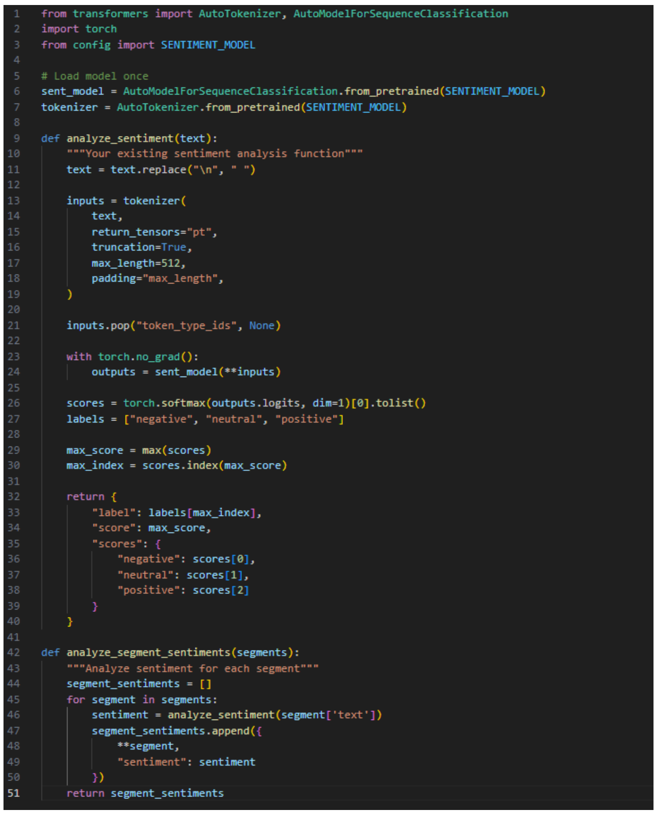
- Remove fillers, normalize punctuation, and structure dialogue.



```python
import re
import nltk
from nltk.tokenize import sent_tokenize

nltk.download('punkt', quiet=True)

def clean_text(text):
    """Clean transcript text - remove fillers, normalize punctuation"""
    fillers = [
    "uh", "um", "uhh", "umm",
    "ah", "er", "eh", "hmm",
    "kinda", "sorta",
    "oh", "ohhhh",
    "huh", "mm-hmm", "mmhm",
    "lmao", "lol"
    ]

    for filler in fillers:
        text = re.sub(r'\b' + filler + r'\b', '', text, flags=re.IGNORECASE)

    text = re.sub(r'\s+', ' ', text)

    text = re.sub(r'\s+([.,!?;])', r'\1', text)
    text = re.sub(r'([.,!?;])(\w)', r'\1 \2', text)

    return text.strip()

def clean_transcript_segments(segments):
    """Clean all transcript segments"""
    cleaned_segments = []
    for segment in segments:
        cleaned_text = clean_text(segment.get('text', ''))
        if cleaned_text:
            cleaned_segments.append({
                **segment,
                'text': cleaned_text
            })
    return cleaned_segments

def format_transcript_for_display(segments):
    """Format transcript for nice display"""
    formatted = []
    for seg in segments:
        formatted.append(f"{seg['speaker']} | {seg['start']:.2f}s → {seg['end']:.2f}s\n{seg['text']}\n")
    return "\n".join(formatted)
```

Figure 25: Filtering the Text

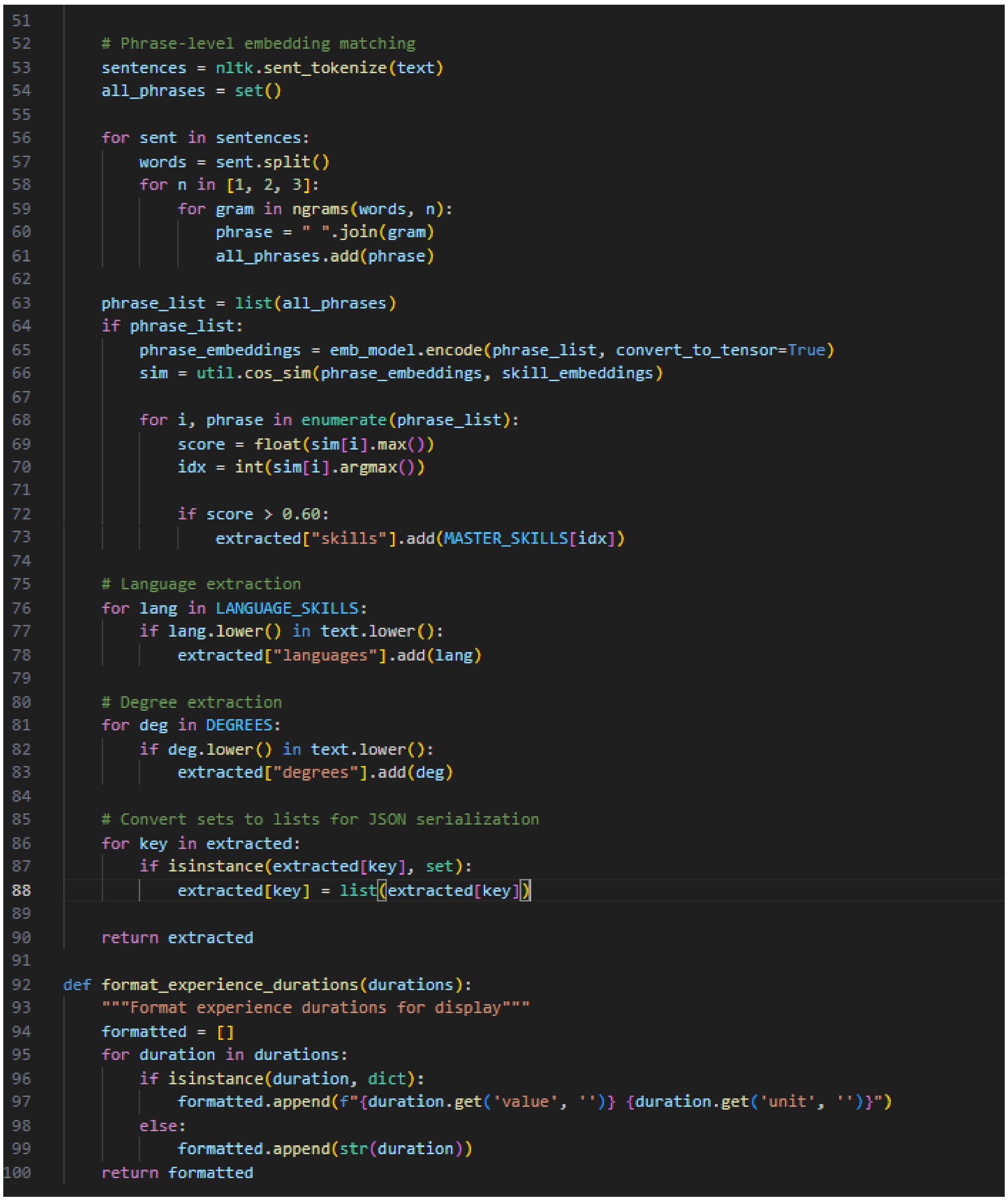
- Perform segment-level and full-transcript sentiment analysis.



```python
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
from config import SENTIMENT_MODEL

# Load model once
sent_model = AutoModelForSequenceClassification.from_pretrained(SENTIMENT_MODEL)
tokenizer = AutoTokenizer.from_pretrained(SENTIMENT_MODEL)

def analyze_sentiment(text):
    """Your existing sentiment analysis function"""
    text = text.replace("\n", " ")

    inputs = tokenizer(
        text,
        return_tensors="pt",
        truncation=True,
        max_length=512,
        padding="max_length",
    )

    inputs.pop("token_type_ids", None)

    with torch.no_grad():
        outputs = sent_model(**inputs)

    scores = torch.softmax(outputs.logits, dim=1)[0].tolist()
    labels = ["negative", "neutral", "positive"]

    max_score = max(scores)
    max_index = scores.index(max_score)

    return {
        "label": labels[max_index],
        "score": max_score,
        "scores": {
            "negative": scores[0],
            "neutral": scores[1],
            "positive": scores[2]
        }
    }

def analyze_segment_sentiments(segments):
    """Analyze sentiment for each segment"""
    segment_sentiments = []
    for segment in segments:
        sentiment = analyze_sentiment(segment['text'])
        segment_sentiments.append({
            **segment,
            "sentiment": sentiment
        })
    return segment_sentiments
```

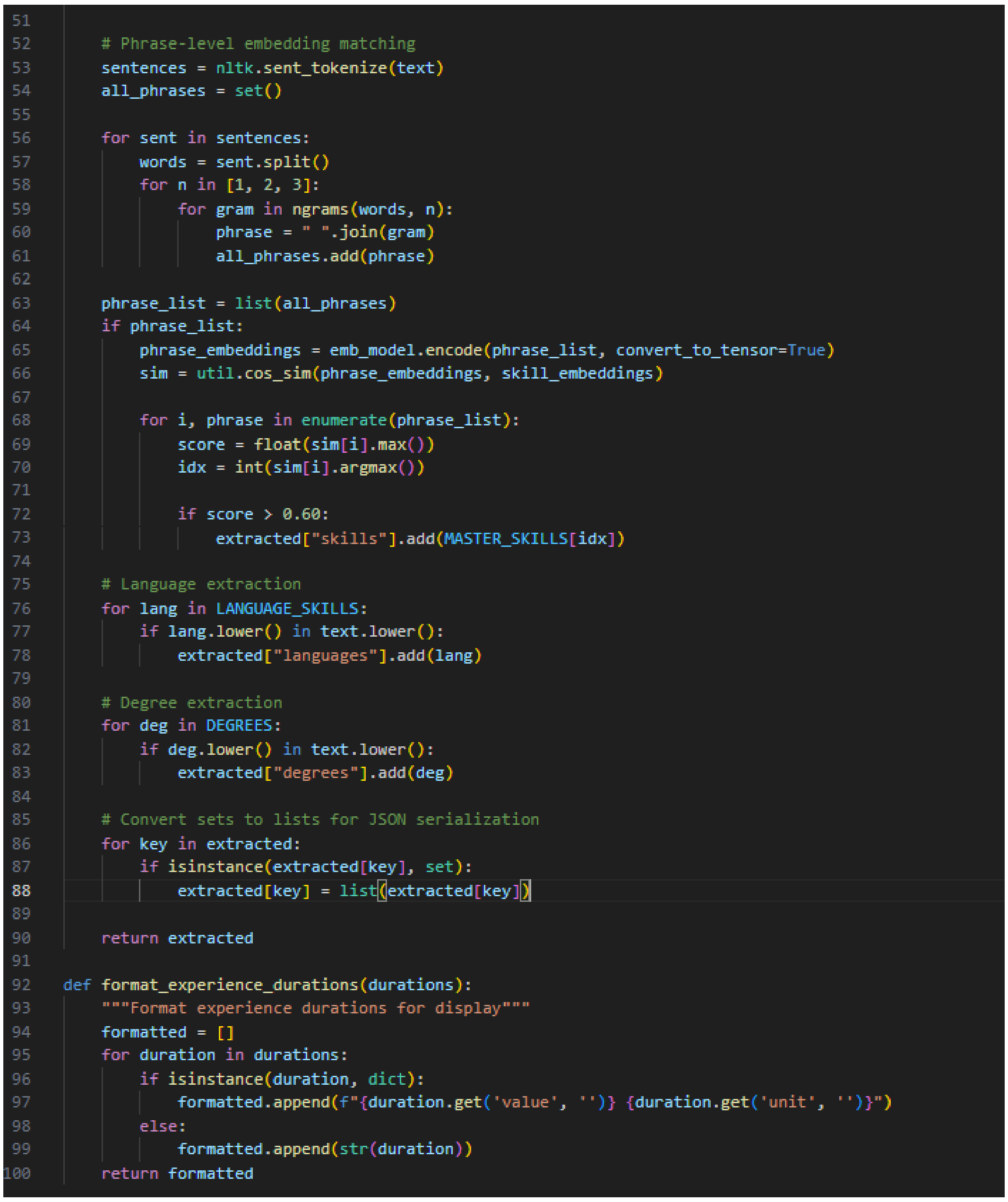
Figure 26: Segmentation

- Use SpaCy + SentenceTransformer to extract:
  – Skills
  – Tools
  – Languages
  – Education
  – Organizations



```python

    # Phrase-level embedding matching
    sentences = nltk.sent_tokenize(text)
    all_phrases = set()

    for sent in sentences:
        words = sent.split()
        for n in [1, 2, 3]:
            for gram in ngrams(words, n):
                phrase = " ".join(gram)
                all_phrases.add(phrase)

    phrase_list = list(all_phrases)
    if phrase_list:
        phrase_embeddings = emb_model.encode(phrase_list, convert_to_tensor=True)
        sim = util.cos_sim(phrase_embeddings, skill_embeddings)

        for i, phrase in enumerate(phrase_list):
            score = float(sim[i].max())
            idx = int(sim[i].argmax())

            if score > 0.60:
                extracted["skills"].add(MASTER_SKILLS[idx])

    # Language extraction
    for lang in LANGUAGE_SKILLS:
        if lang.lower() in text.lower():
            extracted["languages"].add(lang)

    # Degree extraction
    for deg in DEGREES:
        if deg.lower() in text.lower():
            extracted["degrees"].add(deg)

    # Convert sets to lists for JSON serialization
    for key in extracted:
        if isinstance(extracted[key], set):
            extracted[key] = list(extracted[key])

    return extracted

def format_experience_durations(durations):
    """Format experience durations for display"""
    formatted = []
    for duration in durations:
        if isinstance(duration, dict):
            formatted.append(f"{duration.get('value', '')} {duration.get('unit', '')}")
        else:
            formatted.append(str(duration))
    return formatted
```

Figure 27: Skill Extraction



```python

    # Phrase-level embedding matching
    sentences = nltk.sent_tokenize(text)
    all_phrases = set()

    for sent in sentences:
        words = sent.split()
        for n in [1, 2, 3]:
            for gram in ngrams(words, n):
                phrase = " ".join(gram)
                all_phrases.add(phrase)

    phrase_list = list(all_phrases)
    if phrase_list:
        phrase_embeddings = emb_model.encode(phrase_list, convert_to_tensor=True)
        sim = util.cos_sim(phrase_embeddings, skill_embeddings)

        for i, phrase in enumerate(phrase_list):
            score = float(sim[i].max())
            idx = int(sim[i].argmax())

            if score > 0.60:
                extracted["skills"].add(MASTER_SKILLS[idx])

    # Language extraction
    for lang in LANGUAGE_SKILLS:
        if lang.lower() in text.lower():
            extracted["languages"].add(lang)

    # Degree extraction
    for deg in DEGREES:
        if deg.lower() in text.lower():
            extracted["degrees"].add(deg)

    # Convert sets to lists for JSON serialization
    for key in extracted:
        if isinstance(extracted[key], set):
            extracted[key] = list(extracted[key])

    return extracted

def format_experience_durations(durations):
    """Format experience durations for display"""
    formatted = []
    for duration in durations:
        if isinstance(duration, dict):
            formatted.append(f"{duration.get('value', '')} {duration.get('unit', '')}")
        else:
            formatted.append(str(duration))
    return formatted
```

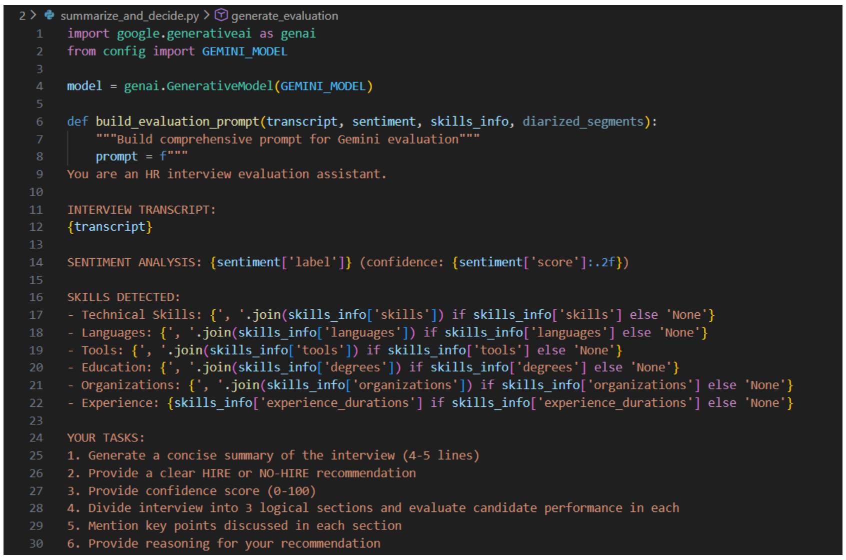
Figure 28: Skill Extraction

MILESTONE 3: AI-Powered HR Evaluation & Decision Engine

This milestone implements the high-level reasoning capabilities that convert raw analysis into meaningful HR insights. Gemini generates summaries, performance breakdowns, and hiring decisions based on structured data.

Activity 3.1: AI Evaluation Prompt Engineering

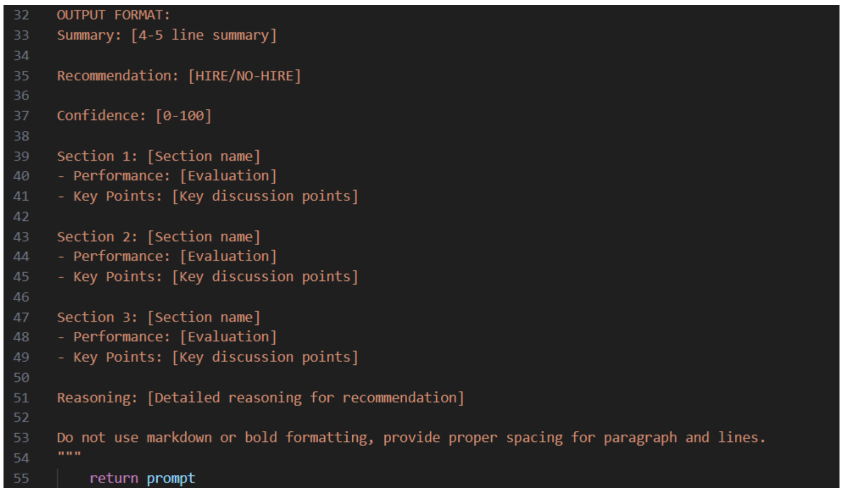
- Build a structured, context-rich evaluation prompt including:
  – Transcript
  – Sentiment result
  – Skills



```
2 > summarize_and_decide.py > generate_evaluation
import google.generativeai as genai
from config import GEMINI_MODEL

model = genai.GenerativeModel(GEMINI_MODEL)

def build_evaluation_prompt(transcript, sentiment, skills_info, diarized_segments):
    """Build comprehensive prompt for Gemini evaluation"""
    prompt = f"""
You are an HR interview evaluation assistant.

INTERVIEW TRANSCRIPT:
{transcript}

SENTIMENT ANALYSIS: {sentiment['label']} (confidence: {sentiment['score']:.2f})

SKILLS DETECTED:
- Technical Skills: {', '.join(skills_info['skills']) if skills_info['skills'] else 'None'}
- Languages: {', '.join(skills_info['languages']) if skills_info['languages'] else 'None'}
- Tools: {', '.join(skills_info['tools']) if skills_info['tools'] else 'None'}
- Education: {', '.join(skills_info['degrees']) if skills_info['degrees'] else 'None'}
- Organizations: {', '.join(skills_info['organizations']) if skills_info['organizations'] else 'None'}
- Experience: {skills_info['experience_durations'] if skills_info['experience_durations'] else 'None'}

YOUR TASKS:
1. Generate a concise summary of the interview (4-5 lines)
2. Provide a clear HIRE or NO-HIRE recommendation
3. Provide confidence score (0-100)
4. Divide interview into 3 logical sections and evaluate candidate performance in each
5. Mention key points discussed in each section
6. Provide reasoning for your recommendation
```

Figure 29: AI Evaluation

- Define output format for Gemini:

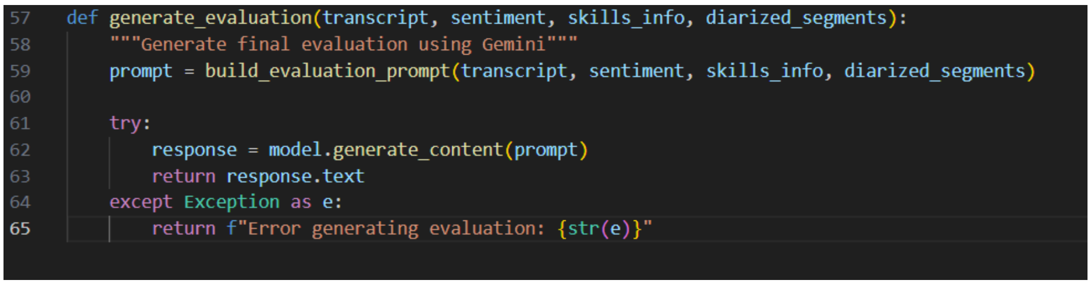


```
OUTPUT FORMAT:
Summary: [4-5 line summary]

Recommendation: [HIRE/NO-HIRE]

Confidence: [0-100]

Section 1: [Section name]
- Performance: [Evaluation]
- Key Points: [Key discussion points]

Section 2: [Section name]
- Performance: [Evaluation]
- Key Points: [Key discussion points]

Section 3: [Section name]
- Performance: [Evaluation]
- Key Points: [Key discussion points]

Reasoning: [Detailed reasoning for recommendation]

Do not use markdown or bold formatting, provide proper spacing for paragraph and lines.
"""
    return prompt
```

Figure 30: Gemini Output Format

Activity 3.2: Generate Final Evaluation with Gemini

- Generate professional summary of the interview.



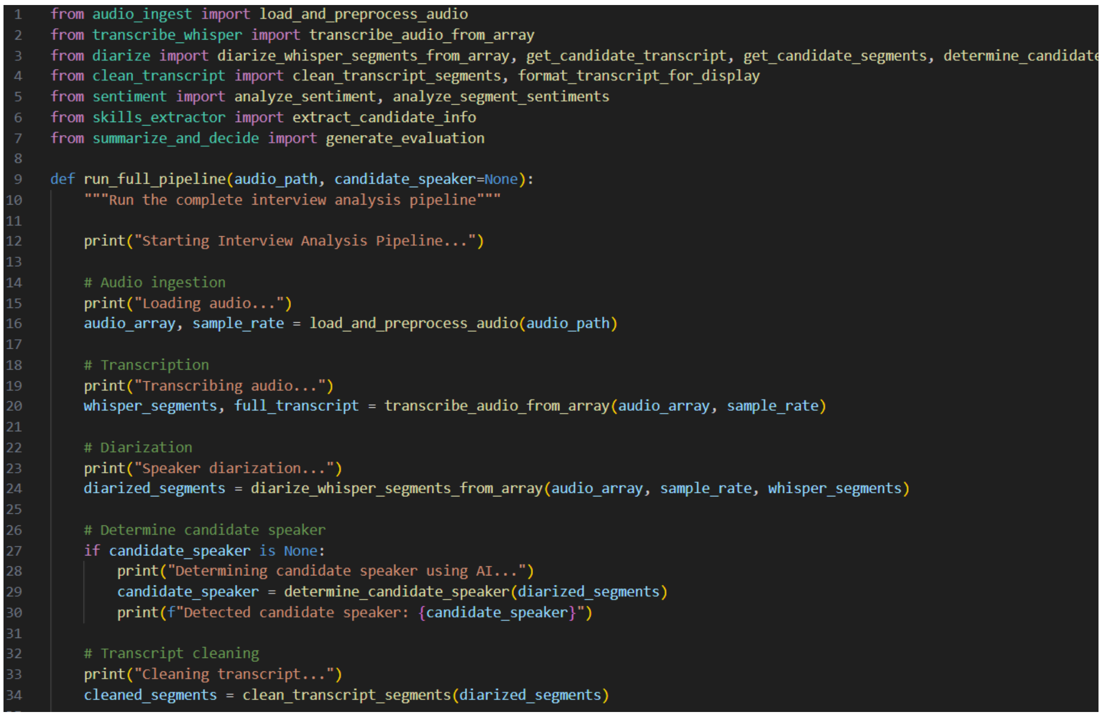
```python
def generate_evaluation(transcript, sentiment, skills_info, diarized_segments):
    """Generate final evaluation using Gemini"""
    prompt = build_evaluation_prompt(transcript, sentiment, skills_info, diarized_segments)

    try:
        response = model.generate_content(prompt)
        return response.text
    except Exception as e:
        return f"Error generating evaluation: {str(e)}"
```

Figure 31: Generating Output with Gemini

- Produce section-wise evaluation across three logical interview phases.
- Generate Hire/No-Hire recommendation with reasoning.
- Store results for reporting and UI display.

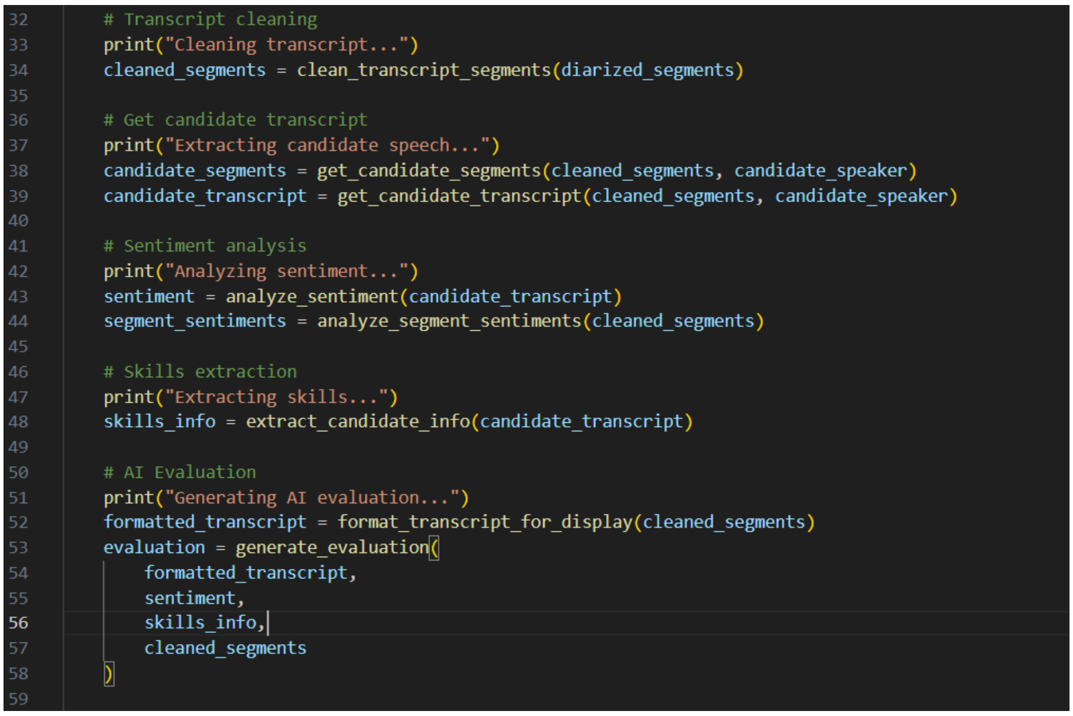
Activity 3.3: Final Pipeline

- Combine sentiment, skills, transcript, and speaker metrics.



```python
from audio_ingest import load_and_preprocess_audio
from transcribe_whisper import transcribe_audio_from_array
from diarize import diarize_whisper_segments_from_array, get_candidate_transcript, get_candidate_segments, determine_candidate
from clean_transcript import clean_transcript_segments, format_transcript_for_display
from sentiment import analyze_sentiment, analyze_segment_sentiments
from skills_extractor import extract_candidate_info
from summarize_and_decide import generate_evaluation

def run_full_pipeline(audio_path, candidate_speaker=None):
    """Run the complete interview analysis pipeline"""

    print("Starting Interview Analysis Pipeline...")

    # Audio ingestion
    print("Loading audio...")
    audio_array, sample_rate = load_and_preprocess_audio(audio_path)

    # Transcription
    print("Transcribing audio...")
    whisper_segments, full_transcript = transcribe_audio_from_array(audio_array, sample_rate)

    # Diarization
    print("Speaker diarization...")
    diarized_segments = diarize_whisper_segments_from_array(audio_array, sample_rate, whisper_segments)

    # Determine candidate speaker
    if candidate_speaker is None:
        print("Determining candidate speaker using AI...")
        candidate_speaker = determine_candidate_speaker(diarized_segments)
        print(f"Detected candidate speaker: {candidate_speaker}")

    # Transcript cleaning
    print("Cleaning transcript...")
    cleaned_segments = clean_transcript_segments(diarized_segments)
```

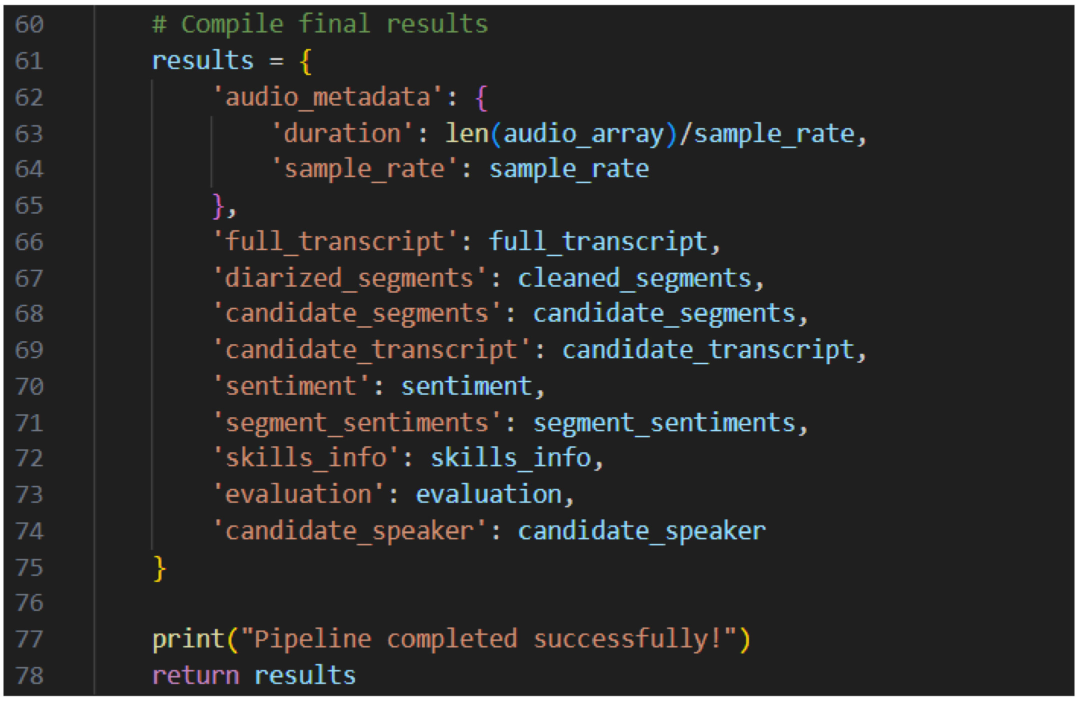
Figure 32: Pipeline



```
    # Transcript cleaning
    print("Cleaning transcript...")
    cleaned_segments = clean_transcript_segments(diarized_segments)

    # Get candidate transcript
    print("Extracting candidate speech...")
    candidate_segments = get_candidate_segments(cleaned_segments, candidate_speaker)
    candidate_transcript = get_candidate_transcript(cleaned_segments, candidate_speaker)

    # Sentiment analysis
    print("Analyzing sentiment...")
    sentiment = analyze_sentiment(candidate_transcript)
    segment_sentiments = analyze_segment_sentiments(cleaned_segments)

    # Skills extraction
    print("Extracting skills...")
    skills_info = extract_candidate_info(candidate_transcript)

    # AI Evaluation
    print("Generating AI evaluation...")
    formatted_transcript = format_transcript_for_display(cleaned_segments)
    evaluation = generate_evaluation(
        formatted_transcript,
        sentiment,
        skills_info,
        cleaned_segments
    )

```

Figure 33: Pipeline

- Compute final metadata including:
  – Duration
  – Segment count
  – Speaker distribution



```
    # Compile final results
    results = {
        'audio_metadata': {
            'duration': len(audio_array)/sample_rate,
            'sample_rate': sample_rate
        },
        'full_transcript': full_transcript,
        'diarized_segments': cleaned_segments,
        'candidate_segments': candidate_segments,
        'candidate_transcript': candidate_transcript,
        'sentiment': sentiment,
        'segment_sentiments': segment_sentiments,
        'skills_info': skills_info,
        'evaluation': evaluation,
        'candidate_speaker': candidate_speaker
    }

    print("Pipeline completed successfully!")
    return results
```

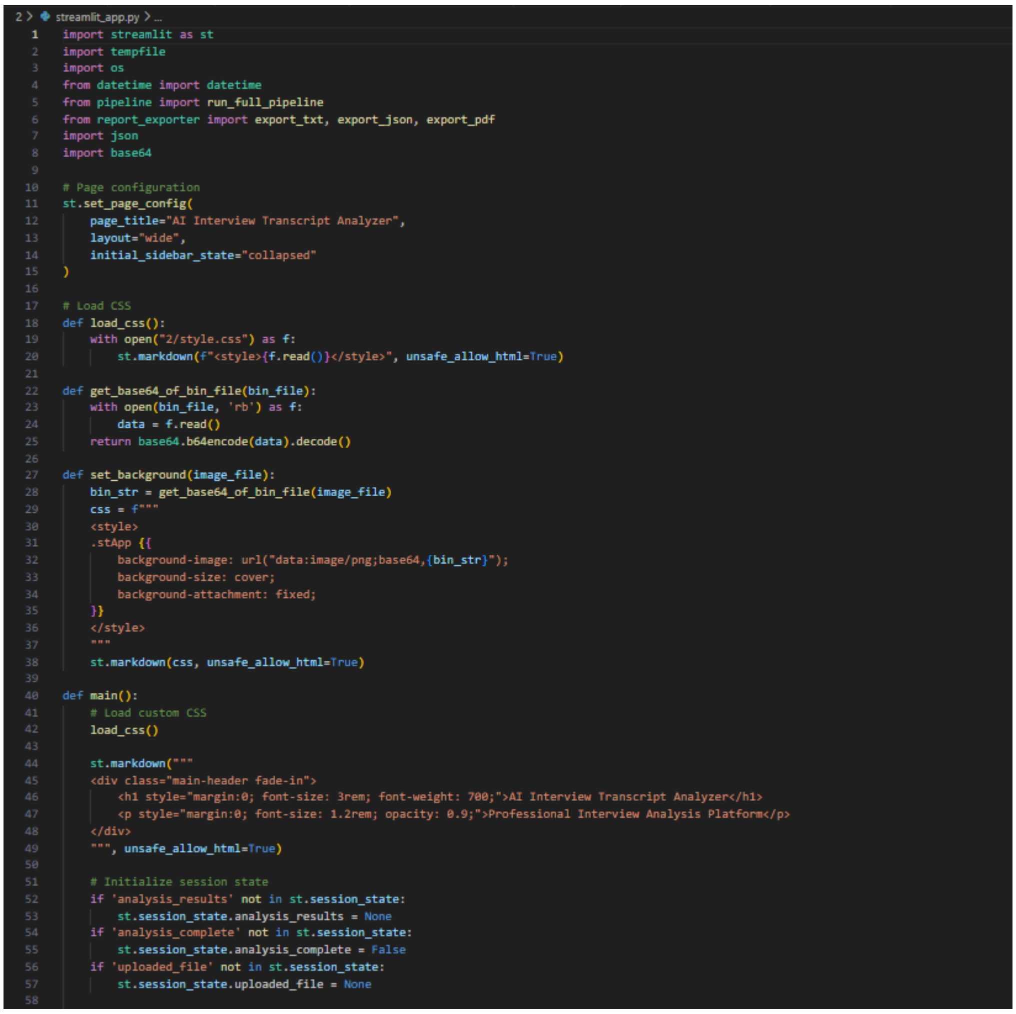
Figure 34: Storing result

MILESTONE 4: Streamlit UI Integration & Application Logic

This milestone focuses on developing a fully interactive, visually refined, and user-friendly platform that brings together every backend component of the AI Interview Transcript Analyzer into a seamless, cohesive user experience. It ensures that the complex processes of audio ingestion, transcription, diarization, sentiment analysis, skills extraction, and AI-driven evaluation are presented to the user in an intuitive, well-structured interface. Through a polished Streamlit UI enhanced with custom styling, responsive design, and multi-tab navigation, this milestone transforms the underlying AI pipeline into a professional, easy-to-use application that allows users to upload interview audio, monitor analysis progress in real time, explore insights, and download reports all within a smooth and engaging workflow.

Activity 4.1: Interactive Layout & Session Management

- Implement session state for:
  – Uploaded file
  – Analysis results
  – UI navigation

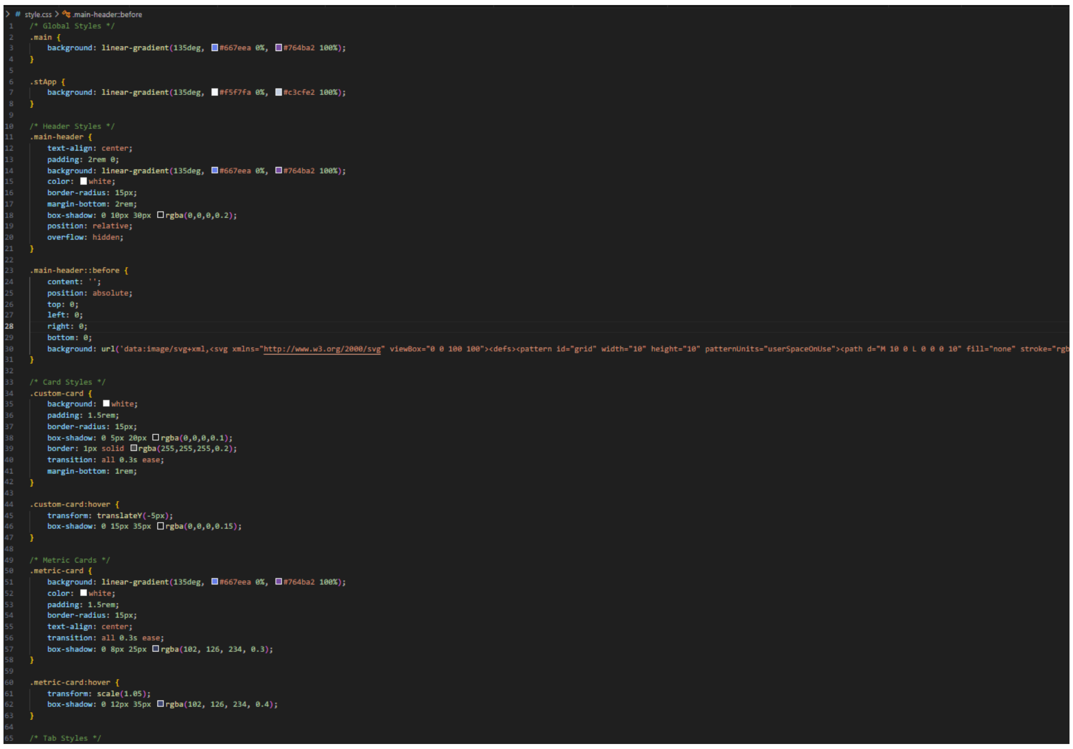


2 > streamlit_app.py > ...

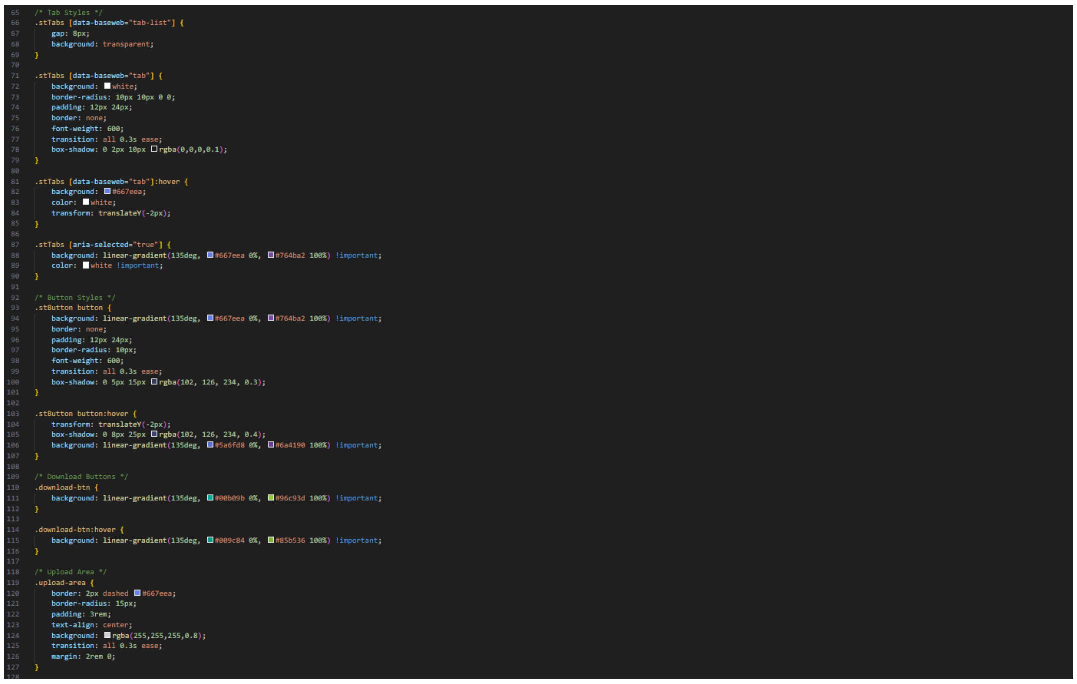
```python
import streamlit as st
import tempfile
import os
from datetime import datetime
from pipeline import run_full_pipeline
from report_exporter import export_txt, export_json, export_pdf
import json
import base64

# Page configuration
st.set_page_config(
    page_title="AI Interview Transcript Analyzer",
    layout="wide",
    initial_sidebar_state="collapsed"
)

# Load CSS
def load_css():
    with open("2/style.css") as f:
        st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)

def get_base64_of_bin_file(bin_file):
    with open(bin_file, 'rb') as f:
        data = f.read()
    return base64.b64encode(data).decode()

def set_background(image_file):
    bin_str = get_base64_of_bin_file(image_file)
    css = f"""
    <style>
    .stApp {{
        background-image: url("data:image/png;base64,{bin_str}");
        background-size: cover;
        background-attachment: fixed;
    }}
    </style>
    """
    st.markdown(css, unsafe_allow_html=True)

def main():
    # Load custom CSS
    load_css()

    st.markdown("""
    <div class="main-header fade-in">
        <h1 style="margin:0; font-size: 3rem; font-weight: 700;">AI Interview Transcript Analyzer</h1>
        <p style="margin:0; font-size: 1.2rem; opacity: 0.9;">Professional Interview Analysis Platform</p>
    </div>
    """, unsafe_allow_html=True)

    # Initialize session state
    if 'analysis_results' not in st.session_state:
        st.session_state.analysis_results = None
    if 'analysis_complete' not in st.session_state:
        st.session_state.analysis_complete = False
    if 'uploaded_file' not in st.session_state:
        st.session_state.uploaded_file = None

```

Figure 35: State Management & Streamlit Configuration

- Configure animations, transitions, and custom CSS.



```css
/* Global Styles */
.main {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
}

.stApp {
    background: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
}

/* Header Styles */
.main-header {
    text-align: center;
    padding: 2rem 0;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    border-radius: 15px;
    margin-bottom: 2rem;
    box-shadow: 0 10px 30px rgba(0,0,0,0.2);
    position: relative;
    overflow: hidden;
}

.main-header::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: url('data:image/svg+xml,<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100"><defs><pattern id="grid" width="10" height="10" patternUnits="userSpaceOnUse"><path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgb
}

/* Card Styles */
.custom-card {
    background: white;
    padding: 1.5rem;
    border-radius: 15px;
    box-shadow: 0 5px 20px rgba(0,0,0,0.1);
    border: 1px solid rgba(255,255,255,0.2);
    transition: all 0.3s ease;
    margin-bottom: 1rem;
}

.custom-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 15px 35px rgba(0,0,0,0.15);
}

/* Metric Cards */
.metric-card {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    padding: 1.5rem;
    border-radius: 15px;
    text-align: center;
    transition: all 0.3s ease;
    box-shadow: 0 8px 25px rgba(102, 126, 234, 0.3);
}

.metric-card:hover {
    transform: scale(1.05);
    box-shadow: 0 12px 35px rgba(102, 126, 234, 0.4);
}

/* Tab Styles */
```

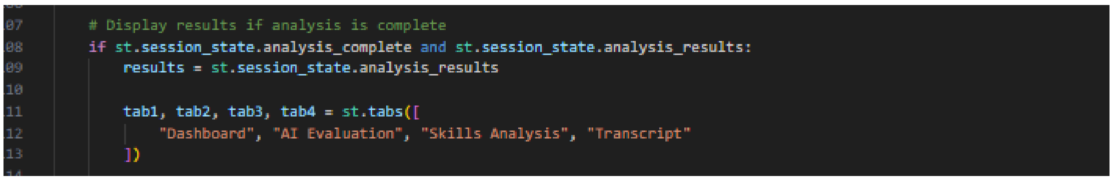


```css
/* Tab Styles */
.stTabs [data-baseweb="tab-list"] {
    gap: 8px;
    background: transparent;
}

.stTabs [data-baseweb="tab"] {
    background: white;
    border-radius: 10px 10px 0 0;
    padding: 12px 24px;
    border: none;
    font-weight: 600;
    transition: all 0.3s ease;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

.stTabs [data-baseweb="tab"]:hover {
    background: #667eea;
    color: white;
    transform: translateY(-2px);
}

.stTabs [aria-selected="true"] {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%) !important;
    color: white !important;
}

/* Button Styles */
.stButton button {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%) !important;
    border: none;
    padding: 12px 24px;
    border-radius: 10px;
    font-weight: 600;
    transition: all 0.3s ease;
    box-shadow: 0 5px 15px rgba(102, 126, 234, 0.3);
}

.stButton button:hover {
    transform: translateY(-2px);
    box-shadow: 0 8px 25px rgba(102, 126, 234, 0.4);
    background: linear-gradient(135deg, #5a6fd8 0%, #6a4190 100%) !important;
}

/* Download Buttons */
.download-btn {
    background: linear-gradient(135deg, #00b09b 0%, #96c93d 100%) !important;
}

.download-btn:hover {
    background: linear-gradient(135deg, #009c84 0%, #85b536 100%) !important;
}

/* Upload Area */
.upload-area {
    border: 2px dashed #667eea;
    border-radius: 15px;
    padding: 3rem;
    text-align: center;
    background: rgba(255,255,255,0.8);
    transition: all 0.3s ease;
    margin: 2rem 0;
}
```

Figure 36: CSS for Enhance Layout

Activity 4.2: Multi-Tab Application Design

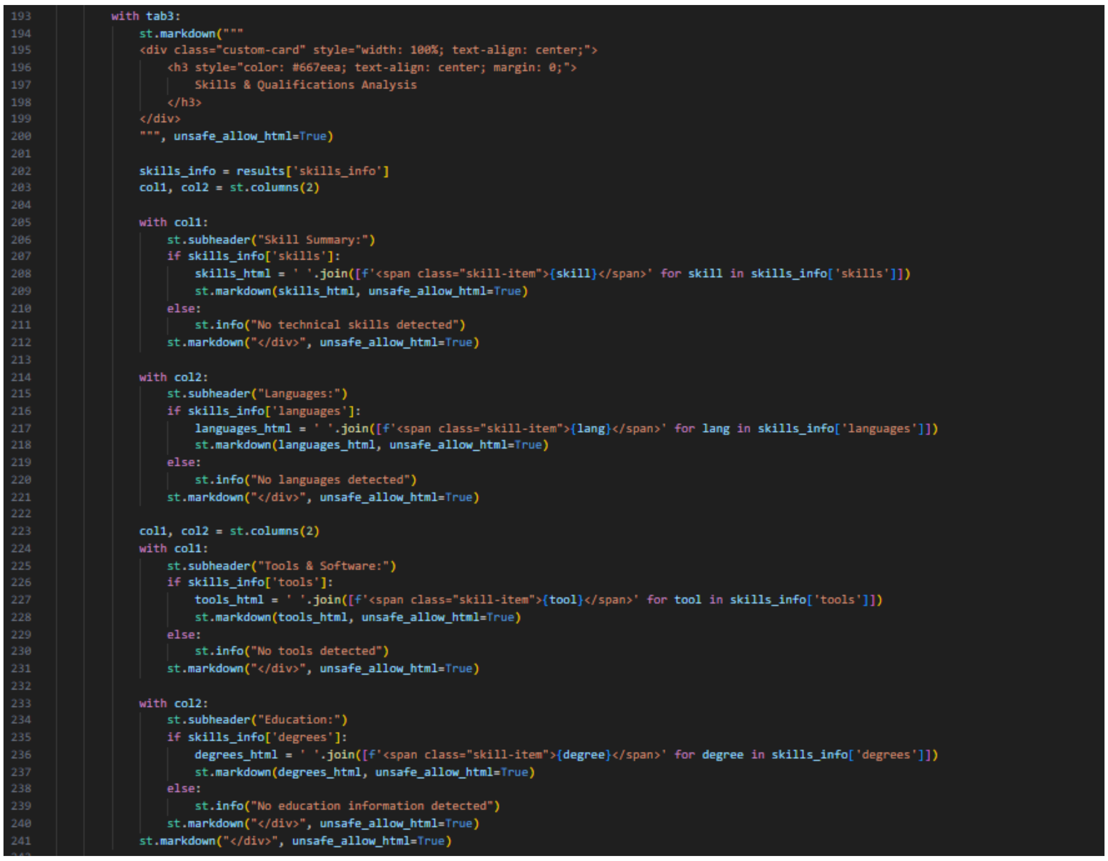
- Tabs implemented:
  1. Dashboard
  2. AI Evaluation
  3. Skills Analysis
  4. Transcript Viewer



```python
    # Display results if analysis is complete
    if st.session_state.analysis_complete and st.session_state.analysis_results:
        results = st.session_state.analysis_results

        tab1, tab2, tab3, tab4 = st.tabs([
            "Dashboard", "AI Evaluation", "Skills Analysis", "Transcript"
        ])
```

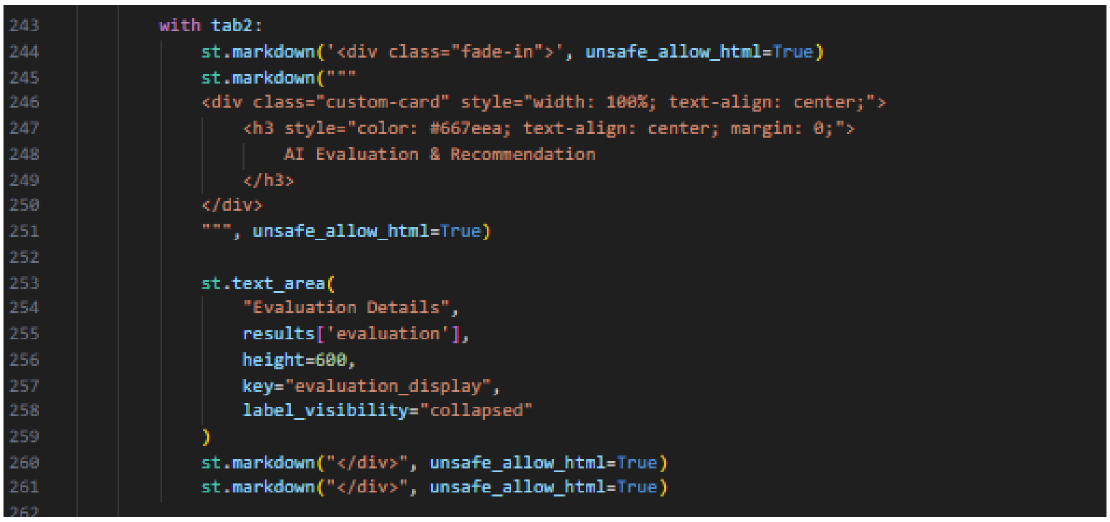
Figure 37: Multi-Tab Option

- Each tab displays structured components and analytics.



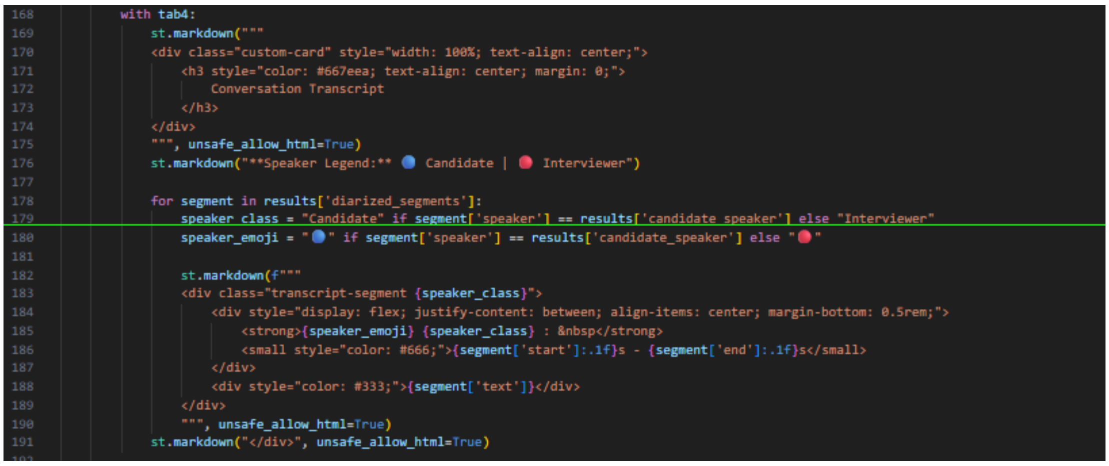
```python
        with tab3:
            st.markdown("""
            <div class="custom-card" style="width: 100%; text-align: center;">
                <h3 style="color: #667eea; text-align: center; margin: 0;">
                    Skills & Qualifications Analysis
                </h3>
            </div>
            """, unsafe_allow_html=True)

            skills_info = results['skills_info']
            col1, col2 = st.columns(2)

            with col1:
                st.subheader("Skill Summary:")
                if skills_info['skills']:
                    skills_html = ' '.join([f'<span class="skill-item">{skill}</span>' for skill in skills_info['skills']])
                    st.markdown(skills_html, unsafe_allow_html=True)
                else:
                    st.info("No technical skills detected")
                st.markdown("</div>", unsafe_allow_html=True)

            with col2:
                st.subheader("Languages:")
                if skills_info['languages']:
                    languages_html = ' '.join([f'<span class="skill-item">{lang}</span>' for lang in skills_info['languages']])
                    st.markdown(languages_html, unsafe_allow_html=True)
                else:
                    st.info("No languages detected")
                st.markdown("</div>", unsafe_allow_html=True)

            col1, col2 = st.columns(2)
            with col1:
                st.subheader("Tools & Software:")
                if skills_info['tools']:
                    tools_html = ' '.join([f'<span class="skill-item">{tool}</span>' for tool in skills_info['tools']])
                    st.markdown(tools_html, unsafe_allow_html=True)
                else:
                    st.info("No tools detected")
                st.markdown("</div>", unsafe_allow_html=True)

            with col2:
                st.subheader("Education:")
                if skills_info['degrees']:
                    degrees_html = ' '.join([f'<span class="skill-item">{degree}</span>' for degree in skills_info['degrees']])
                    st.markdown(degrees_html, unsafe_allow_html=True)
                else:
                    st.info("No education information detected")
                st.markdown("</div>", unsafe_allow_html=True)
            st.markdown("</div>", unsafe_allow_html=True)
```

Figure 38: Tab - Skill & Qualification Analysis



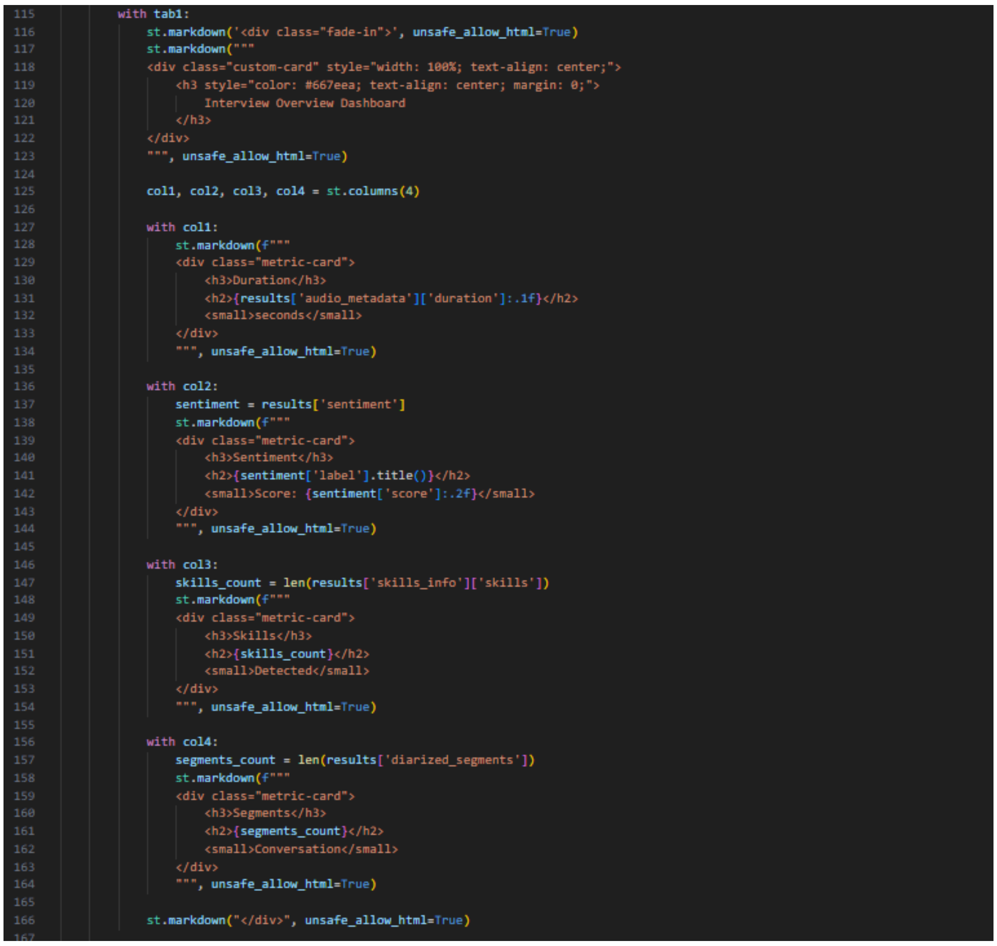
```python
    with tab2:
        st.markdown('<div class="fade-in">', unsafe_allow_html=True)
        st.markdown("""
        <div class="custom-card" style="width: 100%; text-align: center;">
            <h3 style="color: #667eea; text-align: center; margin: 0;">
                AI Evaluation & Recommendation
            </h3>
        </div>
        """, unsafe_allow_html=True)

        st.text_area(
            "Evaluation Details",
            results['evaluation'],
            height=600,
            key="evaluation_display",
            label_visibility="collapsed"
        )
        st.markdown("</div>", unsafe_allow_html=True)
        st.markdown("</div>", unsafe_allow_html=True)
```

Figure 39: Tab - AI Evaluation & Recommendation



```python
    with tab4:
        st.markdown("""
        <div class="custom-card" style="width: 100%; text-align: center;">
            <h3 style="color: #667eea; text-align: center; margin: 0;">
                Conversation Transcript
            </h3>
        </div>
        """, unsafe_allow_html=True)
        st.markdown("**Speaker Legend:** 🔵 Candidate | 🔴 Interviewer")

        for segment in results['diarized_segments']:
            speaker_class = "Candidate" if segment['speaker'] == results['candidate_speaker'] else "Interviewer"
            speaker_emoji = "🔵" if segment['speaker'] == results['candidate_speaker'] else "🔴"

            st.markdown(f"""
            <div class="transcript-segment {speaker_class}">
                <div style="display: flex; justify-content: between; align-items: center; margin-bottom: 0.5rem;">
                    <strong>{speaker_emoji} {speaker_class} : &nbsp</strong>
                    <small style="color: #666;">{segment['start']:.1f}s - {segment['end']:.1f}s</small>
                </div>
                <div style="color: #333;">{segment['text']}</div>
            </div>
            """, unsafe_allow_html=True)
        st.markdown("</div>", unsafe_allow_html=True)
```

Figure 40: Tab - Conversation Transcript

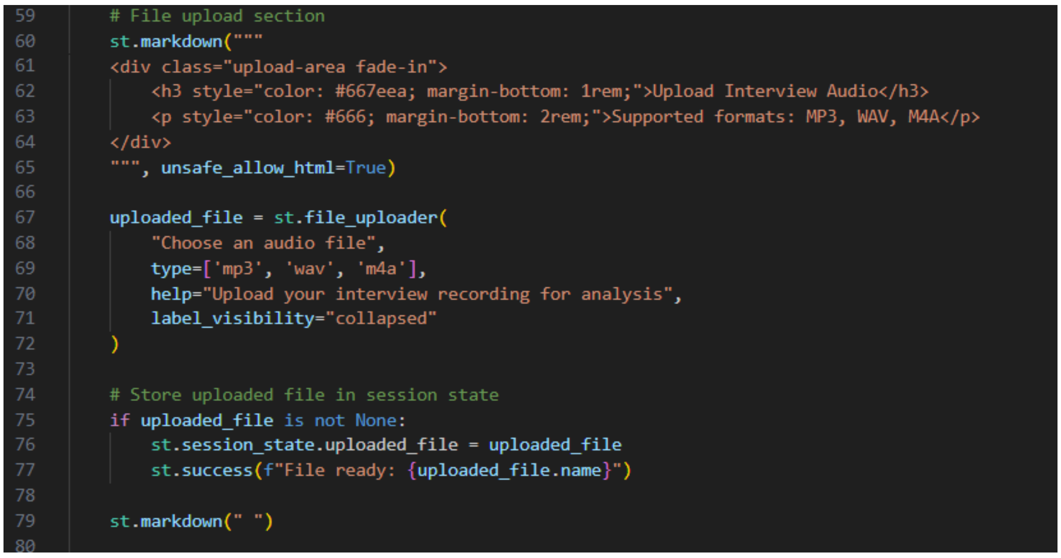


```python
    with tab1:
        st.markdown('<div class="fade-in">', unsafe_allow_html=True)
        st.markdown("""
        <div class="custom-card" style="width: 100%; text-align: center;">
            <h3 style="color: #667eea; text-align: center; margin: 0;">
                Interview Overview Dashboard
            </h3>
        </div>
        """, unsafe_allow_html=True)

        col1, col2, col3, col4 = st.columns(4)

        with col1:
            st.markdown(f"""
            <div class="metric-card">
                <h3>Duration</h3>
                <h2>{results['audio_metadata']['duration']:.1f}</h2>
                <small>seconds</small>
            </div>
            """, unsafe_allow_html=True)

        with col2:
            sentiment = results['sentiment']
            st.markdown(f"""
            <div class="metric-card">
                <h3>Sentiment</h3>
                <h2>{sentiment['label'].title()}</h2>
                <small>Score: {sentiment['score']:.2f}</small>
            </div>
            """, unsafe_allow_html=True)

        with col3:
            skills_count = len(results['skills_info']['skills'])
            st.markdown(f"""
            <div class="metric-card">
                <h3>Skills</h3>
                <h2>{skills_count}</h2>
                <small>Detected</small>
            </div>
            """, unsafe_allow_html=True)

        with col4:
            segments_count = len(results['diarized_segments'])
            st.markdown(f"""
            <div class="metric-card">
                <h3>Segments</h3>
                <h2>{segments_count}</h2>
                <small>Conversation</small>
            </div>
            """, unsafe_allow_html=True)

        st.markdown("</div>", unsafe_allow_html=True)
```

Figure 41: Tab - Interview Overview Dashboard

Activity 4.3: Audio Upload & Processing Workflow

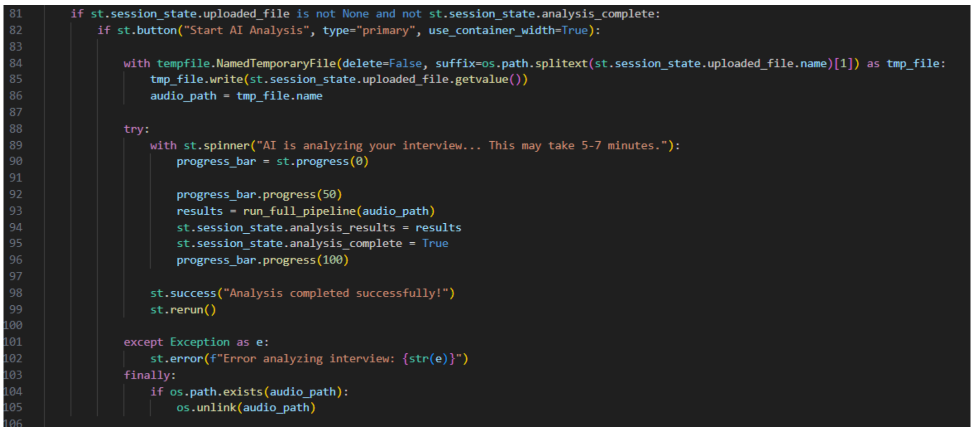
- Implement drag-and-drop audio uploader.



```python
    # File upload section
    st.markdown("""
    <div class="upload-area fade-in">
        <h3 style="color: #667eea; margin-bottom: 1rem;">Upload Interview Audio</h3>
        <p style="color: #666; margin-bottom: 2rem;">Supported formats: MP3, WAV, M4A</p>
    </div>
    """, unsafe_allow_html=True)

    uploaded_file = st.file_uploader(
        "Choose an audio file",
        type=['mp3', 'wav', 'm4a'],
        help="Upload your interview recording for analysis",
        label_visibility="collapsed"
    )

    # Store uploaded file in session state
    if uploaded_file is not None:
        st.session_state.uploaded_file = uploaded_file
        st.success(f"File ready: {uploaded_file.name}")

    st.markdown(" ")
```

Figure 42: Uploading Audio

- Provide progress indicators (0–100%) during analysis with display success or error messages.

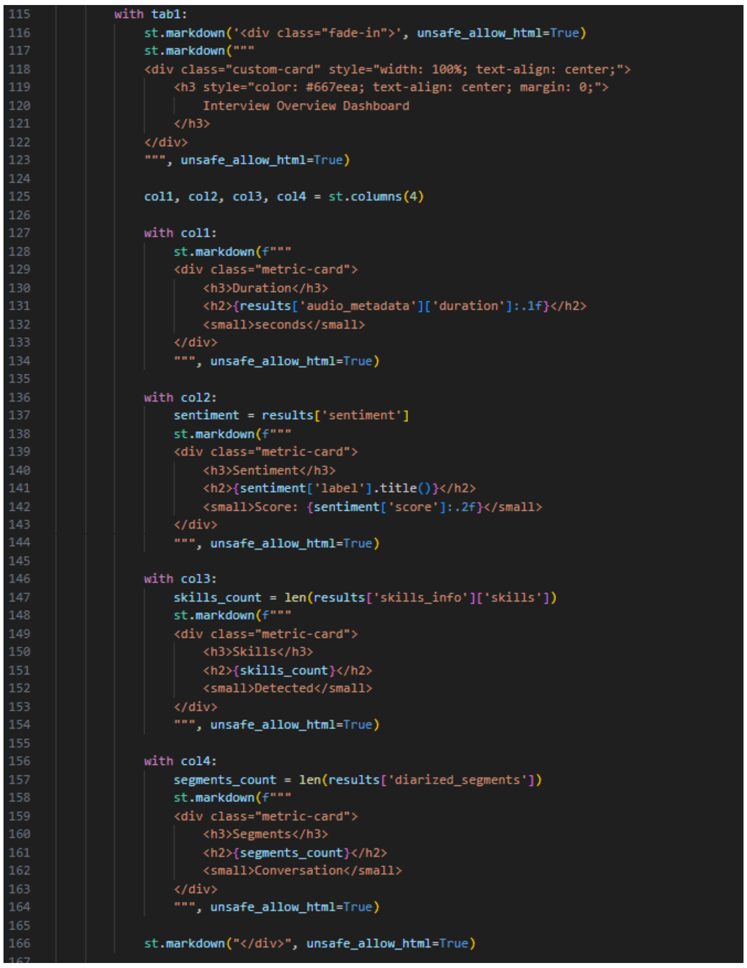


```python
    if st.session_state.uploaded_file is not None and not st.session_state.analysis_complete:
        if st.button("Start AI Analysis", type="primary", use_container_width=True):

            with tempfile.NamedTemporaryFile(delete=False, suffix=os.path.splitext(st.session_state.uploaded_file.name)[1]) as tmp_file:
                tmp_file.write(st.session_state.uploaded_file.getvalue())
                audio_path = tmp_file.name

            try:
                with st.spinner("AI is analyzing your interview... This may take 5-7 minutes."):
                    progress_bar = st.progress(0)

                    progress_bar.progress(50)
                    results = run_full_pipeline(audio_path)
                    st.session_state.analysis_results = results
                    st.session_state.analysis_complete = True
                    progress_bar.progress(100)

                st.success("Analysis completed successfully!")
                st.rerun()

            except Exception as e:
                st.error(f"Error analyzing interview: {str(e)}")
            finally:
                if os.path.exists(audio_path):
                    os.unlink(audio_path)
```

Figure 43: Progress Bar

Activity 4.4: Skills, Sentiment & Transcript Visualization

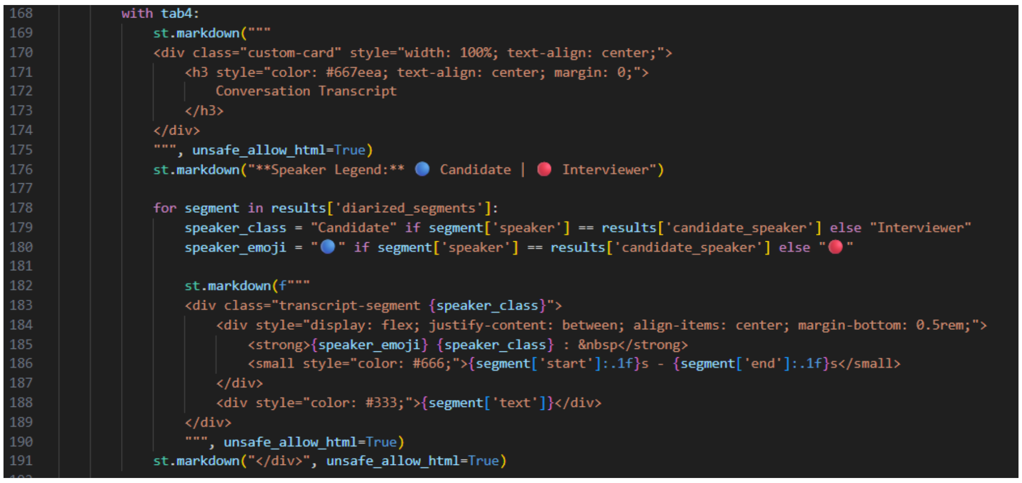
- Skill and other metrics display with custom styling.



```python
        with tab1:
            st.markdown('<div class="fade-in">', unsafe_allow_html=True)
            st.markdown("""
            <div class="custom-card" style="width: 100%; text-align: center;">
                <h3 style="color: #667eea; text-align: center; margin: 0;">
                    Interview Overview Dashboard
                </h3>
            </div>
            """, unsafe_allow_html=True)

            col1, col2, col3, col4 = st.columns(4)

            with col1:
                st.markdown(f"""
                <div class="metric-card">
                    <h3>Duration</h3>
                    <h2>{results['audio_metadata']['duration']:.1f}</h2>
                    <small>seconds</small>
                </div>
                """, unsafe_allow_html=True)

            with col2:
                sentiment = results['sentiment']
                st.markdown(f"""
                <div class="metric-card">
                    <h3>Sentiment</h3>
                    <h2>{sentiment['label'].title()}</h2>
                    <small>Score: {sentiment['score']:.2f}</small>
                </div>
                """, unsafe_allow_html=True)

            with col3:
                skills_count = len(results['skills_info']['skills'])
                st.markdown(f"""
                <div class="metric-card">
                    <h3>Skills</h3>
                    <h2>{skills_count}</h2>
                    <small>Detected</small>
                </div>
                """, unsafe_allow_html=True)

            with col4:
                segments_count = len(results['diarized_segments'])
                st.markdown(f"""
                <div class="metric-card">
                    <h3>Segments</h3>
                    <h2>{segments_count}</h2>
                    <small>Conversation</small>
                </div>
                """, unsafe_allow_html=True)

            st.markdown("</div>", unsafe_allow_html=True)
```

Figure 44: Visualization of tabs

- Transcript with speaker colour coding.

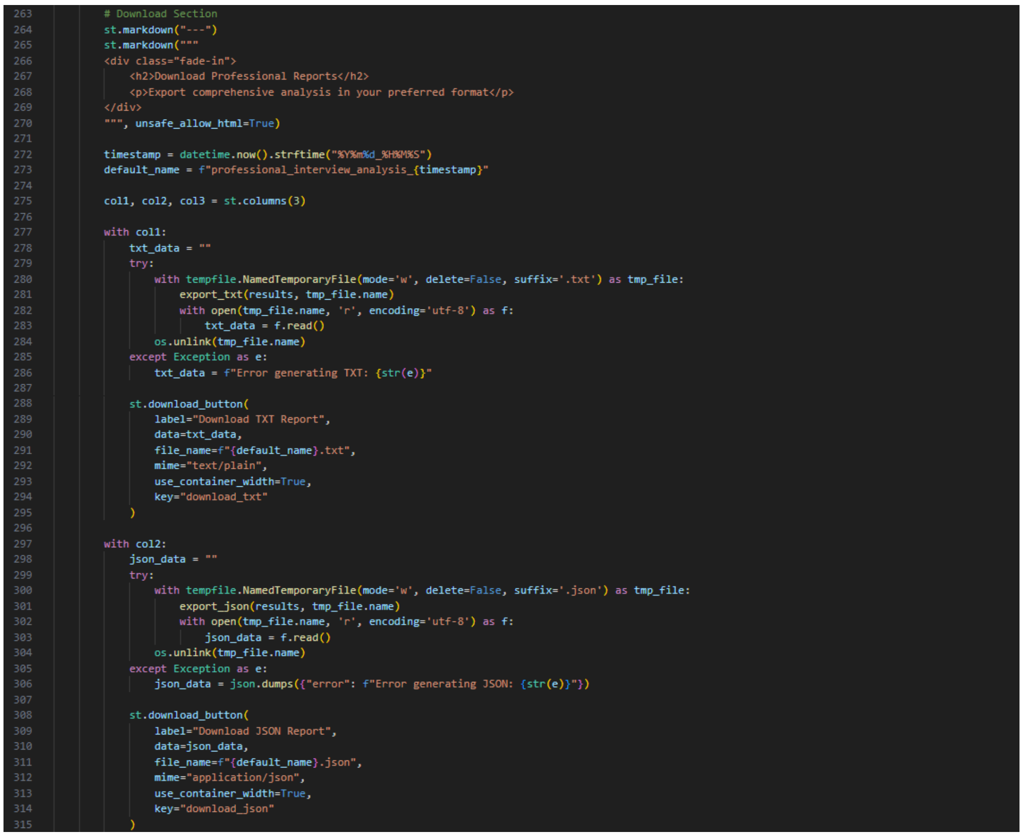


```python
    with tab4:
        st.markdown("""
        <div class="custom-card" style="width: 100%; text-align: center;">
            <h3 style="color: #667eea; text-align: center; margin: 0;">
                Conversation Transcript
            </h3>
        </div>
        """, unsafe_allow_html=True)
        st.markdown("**Speaker Legend:** 🔵 Candidate | 🔴 Interviewer")

        for segment in results['diarized_segments']:
            speaker_class = "Candidate" if segment['speaker'] == results['candidate_speaker'] else "Interviewer"
            speaker_emoji = "🔵" if segment['speaker'] == results['candidate_speaker'] else "🔴"

            st.markdown(f"""
            <div class="transcript-segment {speaker_class}">
                <div style="display: flex; justify-content: between; align-items: center; margin-bottom: 0.5rem;">
                    <strong>{speaker_emoji} {speaker_class} : &nbsp</strong>
                    <small style="color: #666;">{segment['start']:.1f}s - {segment['end']:.1f}s</small>
                </div>
                <div style="color: #333;">{segment['text']}</div>
            </div>
            """, unsafe_allow_html=True)
        st.markdown("</div>", unsafe_allow_html=True)
```

Figure 45: Custom colour for Speakers

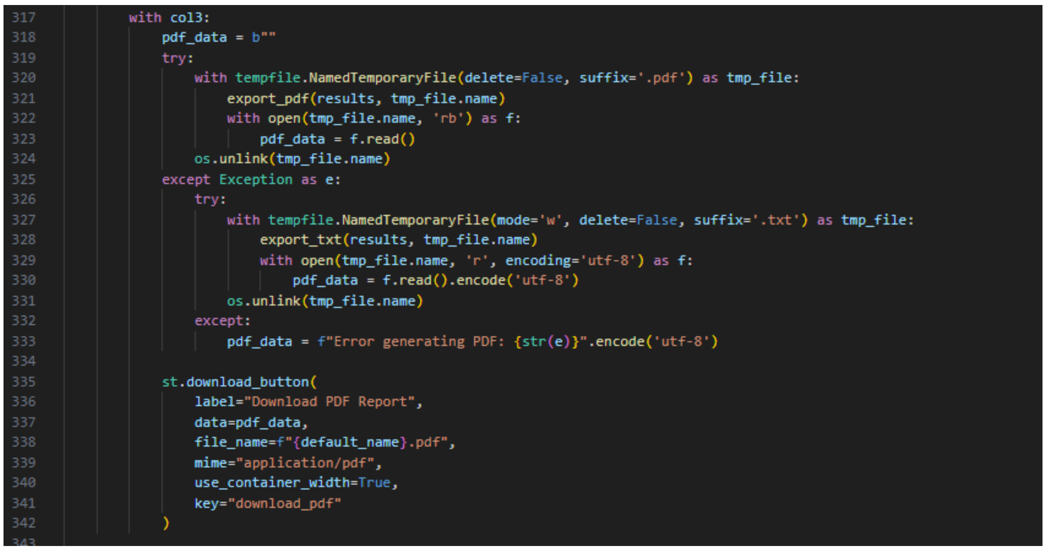
Activity 4.5: Report Export System

- Generate downloadable reports in:
  – TXT
  – JSON
  – PDF



```python
        # Download Section
        st.markdown("---")
        st.markdown("""
        <div class="fade-in">
            <h2>Download Professional Reports</h2>
            <p>Export comprehensive analysis in your preferred format</p>
        </div>
        """, unsafe_allow_html=True)

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        default_name = f"professional_interview_analysis_{timestamp}"

        col1, col2, col3 = st.columns(3)

        with col1:
            txt_data = ""
            try:
                with tempfile.NamedTemporaryFile(mode='w', delete=False, suffix='.txt') as tmp_file:
                    export_txt(results, tmp_file.name)
                    with open(tmp_file.name, 'r', encoding='utf-8') as f:
                        txt_data = f.read()
                os.unlink(tmp_file.name)
            except Exception as e:
                txt_data = f"Error generating TXT: {str(e)}"

            st.download_button(
                label="Download TXT Report",
                data=txt_data,
                file_name=f"{default_name}.txt",
                mime="text/plain",
                use_container_width=True,
                key="download_txt"
            )

        with col2:
            json_data = ""
            try:
                with tempfile.NamedTemporaryFile(mode='w', delete=False, suffix='.json') as tmp_file:
                    export_json(results, tmp_file.name)
                    with open(tmp_file.name, 'r', encoding='utf-8') as f:
                        json_data = f.read()
                os.unlink(tmp_file.name)
            except Exception as e:
                json_data = json.dumps({"error": f"Error generating JSON: {str(e)}"})

            st.download_button(
                label="Download JSON Report",
                data=json_data,
                file_name=f"{default_name}.json",
                mime="application/json",
                use_container_width=True,
                key="download_json"
            )
```

Figure 46: Download button for each format



```python
        with col3:
            pdf_data = b""
            try:
                with tempfile.NamedTemporaryFile(delete=False, suffix='.pdf') as tmp_file:
                    export_pdf(results, tmp_file.name)
                    with open(tmp_file.name, 'rb') as f:
                        pdf_data = f.read()
                os.unlink(tmp_file.name)
            except Exception as e:
                try:
                    with tempfile.NamedTemporaryFile(mode='w', delete=False, suffix='.txt') as tmp_file:
                        export_txt(results, tmp_file.name)
                        with open(tmp_file.name, 'r', encoding='utf-8') as f:
                            pdf_data = f.read().encode('utf-8')
                    os.unlink(tmp_file.name)
                except:
                    pdf_data = f"Error generating PDF: {str(e)}".encode('utf-8')

            st.download_button(
                label="Download PDF Report",
                data=pdf_data,
                file_name=f"{default_name}.pdf",
                mime="application/pdf",
                use_container_width=True,
                key="download_pdf"
            )
```

Figure 47: Download Button

MILESTONE 5: Testing, Optimization & Deployment

This milestone ensures that the AI Interview Transcript Analyzer delivers a polished, intuitive, and fully functional user experience. The focus is on visually validating UI components, testing the display of interview insights, and confirming that users can seamlessly upload audio, view analysis results, and download reports. Performance checks and deployment finalization are handled afterward.

Activity 5.1: UI Display Testing & Result Verification

- **Validate Home Page UI and Feature Cards**

  Verify that feature sections (Transcription, Diarization, Sentiment, Skills) render with proper styling.

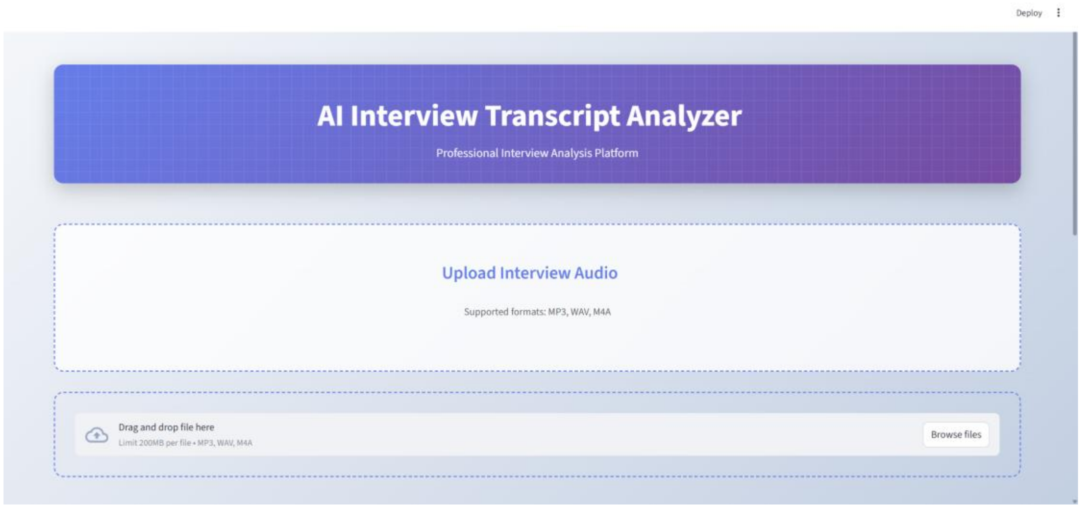

Deploy
AI Interview Transcript Analyzer
Professional Interview Analysis Platform
Upload Interview Audio
Supported formats: MP3, WAV, M4A
Drag and drop file here
Limit 200MB per file • MP3, WAV, M4A
Browse files

Figure 48: Streamlit Home Page

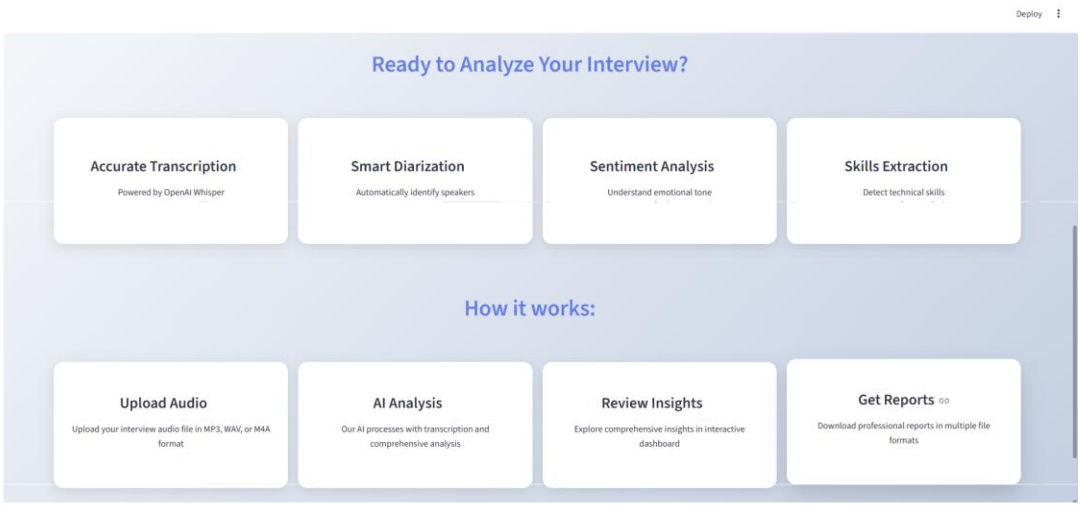
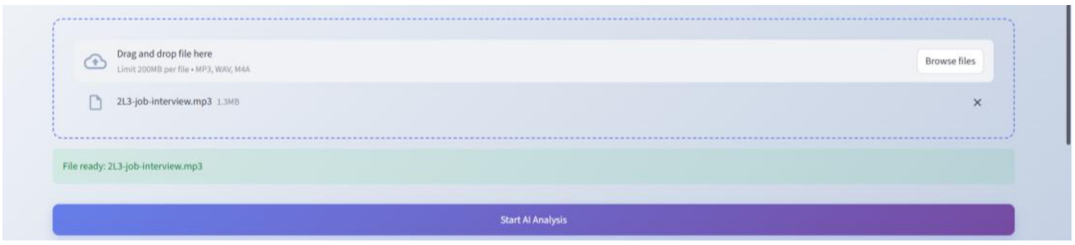

Deploy
Ready to Analyze Your Interview?
Accurate Transcription
Powered by OpenAI Whisper
Smart Diarization
Automatically identify speakers
Sentiment Analysis
Understand emotional tone
Skills Extraction
Detect technical skills
How it works:
Upload Audio
Upload your interview audio file in MP3, WAV, or M4A format
AI Analysis
Our AI processes with transcription and comprehensive analysis
Review Insights
Explore comprehensive insights in interactive dashboard
Get Reports
Download professional reports in multiple file formats

Figure 49: Streamlit About Section

- **Test Audio Upload Workflow**

  Upload MP3/WAV/M4A files and confirm success messages appear.

  Verify file handler stores uploaded audio in session state.


Drag and drop file here
Limit 200MB per file • MP3, WAV, M4A
Browse files
2L3-job-interview.mp3 1.3MB
Start AI Analysis

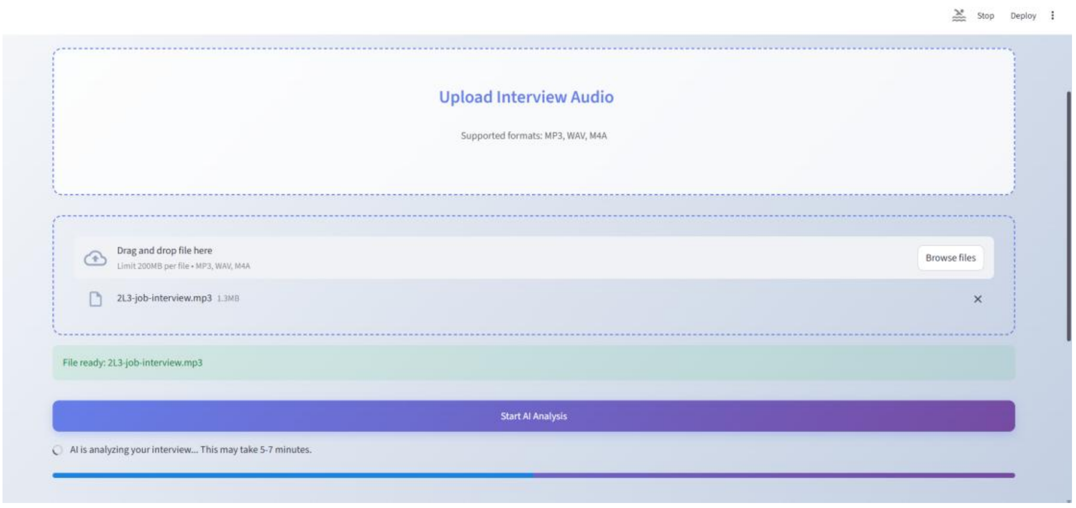
File ready: 2L3-job-interview.mp3

Figure 50: Uploading Section

- **Verify Processing Feedback & Progress Bar**

  Validate progress indicators during analysis (0% → 50% → 100%).

  Confirm loading animations and spinner messages appear.


Stop
Deploy
Upload Interview Audio
Supported formats: MP3, WAV, M4A
Drag and drop file here
Limit 200MB per file • MP3, WAV, M4A
Browse files
2L3-job-interview.mp3 1.3MB
Start AI Analysis

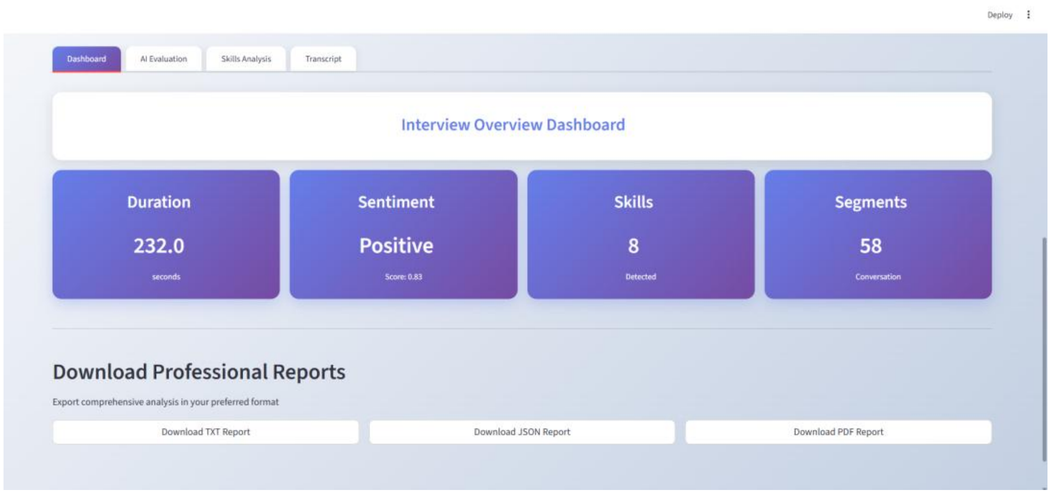
File ready: 2L3-job-interview.mp3

AI is analyzing your interview... This may take 5-7 minutes.

Figure 51: Progress Bar

- **Dashboard Metrics Display Testing**

  Check that Duration, Sentiment Score, Skill Count, and Segments display correctly in metric cards.



Interview Overview Dashboard

Duration
232.0
seconds

Sentiment
Positive
Score: 0.83

Skills
8
Detected

Segments
58
Conversation

Download Professional Reports

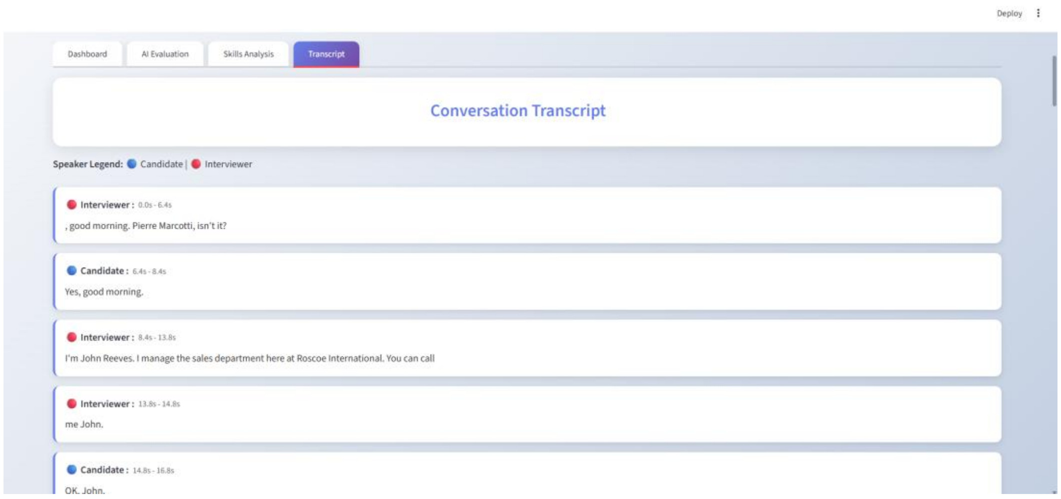
Export comprehensive analysis in your preferred format

Figure 52: Dashboard

- **Transcript Viewer Validation**

  Confirm diarized segments render with Candidate/Interviewer colour coding.

  Validate timestamp formatting and segment spacing.



Conversation Transcript

Speaker Legend: 🔵 Candidate | 🔴 Interviewer

🔴 Interviewer : 0.0s - 6.4s
, good morning. Pierre Marcotti, isn't it?

🔵 Candidate : 6.4s - 8.4s
Yes, good morning.

🔴 Interviewer : 8.4s - 13.8s
I'm John Reeves. I manage the sales department here at Roscoe International. You can call

🔴 Interviewer : 13.8s - 14.8s
me John.

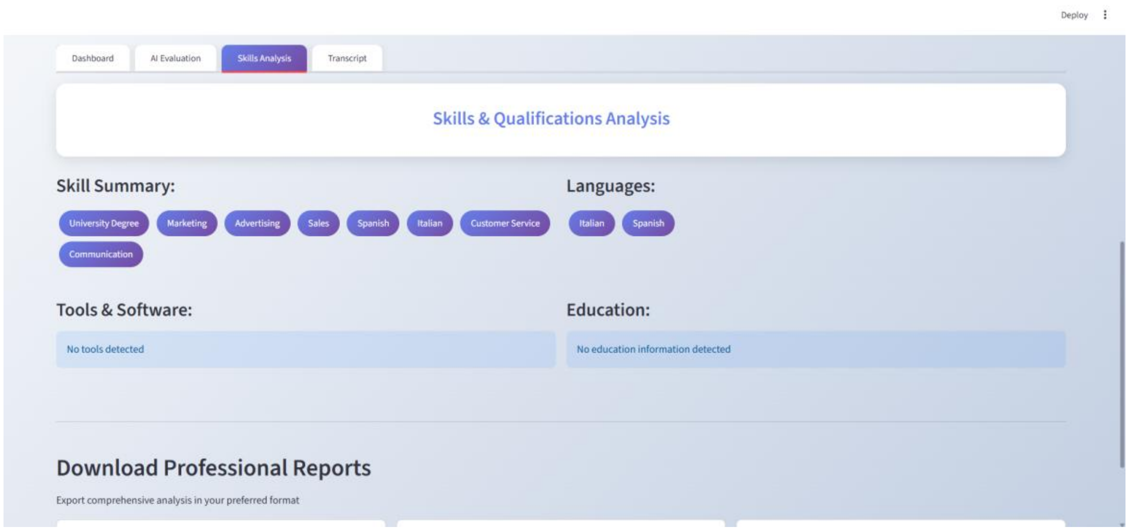
🔵 Candidate : 14.8s - 16.8s
OK. John.

Figure 53: Transcript

- **Skills Analysis Tab UI Testing**

  Check tag-based display for Skills, Languages, Tools, and Education.

  Validate responsive grid layout for different screen sizes.



Skills & Qualifications Analysis

Skill Summary:

University Degree · Marketing · Advertising · Sales · Spanish · Italian · Customer Service · Communication

Languages:

Italian · Spanish

Tools & Software:

No tools detected

Education:

No education information detected

Download Professional Reports

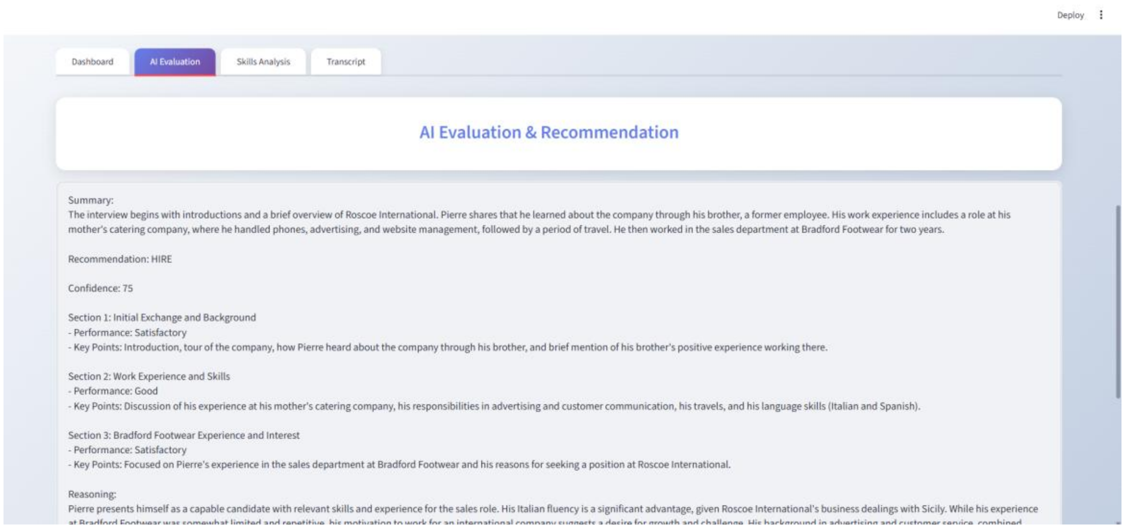
Export comprehensive analysis in your preferred format

Figure 54: Skill Extractor

- **AI Evaluation Display Testing**

  Verify evaluation text loads fully inside scrollable text area.

  Check formatting of Summary, Recommendation, Sections, and Reasoning.



AI Evaluation & Recommendation

Summary:
The interview begins with introductions and a brief overview of Roscoe International. Pierre shares that he learned about the company through his brother, a former employee. His work experience includes a role at his mother's catering company, where he handled phones, advertising, and website management, followed by a period of travel. He then worked in the sales department at Bradford Footwear for two years.

Recommendation: HIRE

Confidence: 75

Section 1: Initial Exchange and Background
- Performance: Satisfactory
- Key Points: Introduction, tour of the company, how Pierre heard about the company through his brother, and brief mention of his brother's positive experience working there.

Section 2: Work Experience and Skills
- Performance: Good
- Key Points: Discussion of his experience at his mother's catering company, his responsibilities in advertising and customer communication, his travels, and his language skills (Italian and Spanish).

Section 3: Bradford Footwear Experience and Interest
- Performance: Satisfactory
- Key Points: Focused on Pierre's experience in the sales department at Bradford Footwear and his reasons for seeking a position at Roscoe International.

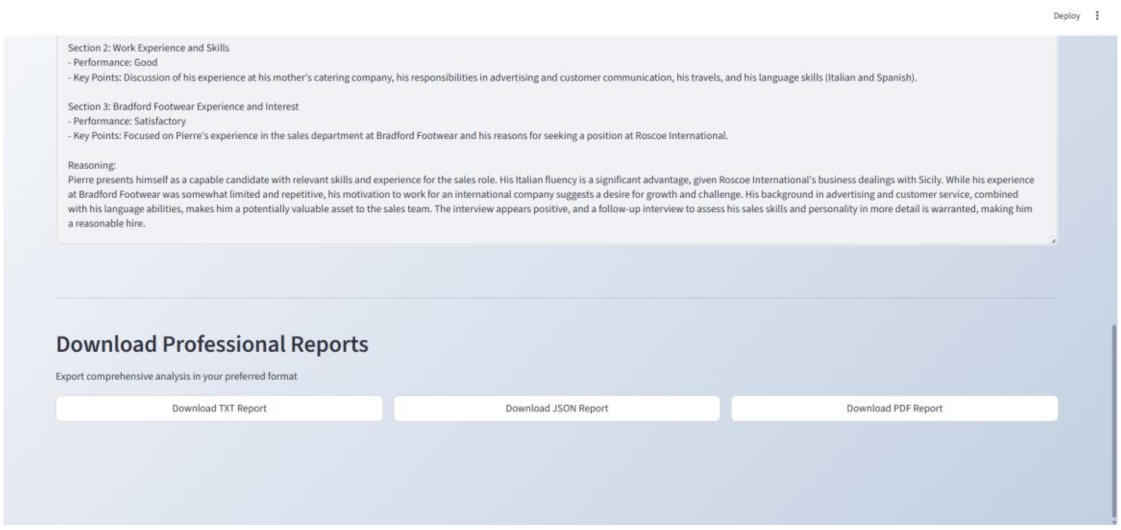
Reasoning:
Pierre presents himself as a capable candidate with relevant skills and experience for the sales role. His Italian fluency is a significant advantage, given Roscoe International's business dealings with Sicily. While his experience at Bradford Footwear was somewhat limited and repetitive, his motivation to work for an international company suggests a desire for growth and challenge. His background in advertising and customer service, combined

Figure 55: AI Evaluation

- **Download Buttons Testing**

  Ensure TXT, JSON, PDF buttons appear correctly with icons/styling.

  Verify downloaded files have correct names and valid content.



Section 2: Work Experience and Skills
- Performance: Good
- Key Points: Discussion of his experience at his mother's catering company, his responsibilities in advertising and customer communication, his travels, and his language skills (Italian and Spanish).

Section 3: Bradford Footwear Experience and Interest
- Performance: Satisfactory
- Key Points: Focused on Pierre's experience in the sales department at Bradford Footwear and his reasons for seeking a position at Roscoe International.

Reasoning:
Pierre presents himself as a capable candidate with relevant skills and experience for the sales role. His Italian fluency is a significant advantage, given Roscoe International's business dealings with Sicily. While his experience at Bradford Footwear was somewhat limited and repetitive, his motivation to work for an international company suggests a desire for growth and challenge. His background in advertising and customer service, combined with his language abilities, makes him a potentially valuable asset to the sales team. The interview appears positive, and a follow-up interview to assess his sales skills and personality in more detail is warranted, making him a reasonable hire.

Download Professional Reports

Export comprehensive analysis in your preferred format

Figure 56: Download Buttons

Activity 5.2: Deployment Preparation & Validation

- **Prepare Streamlit for Production Deployment**

  Configured Streamlit page settings, optimized custom CSS loading, and validated UI components to ensure smooth rendering in the deployed environment.

- **Verify Environment Variables and Model Access**

  Confirmed the GEMINI_API_KEY and all model configurations load correctly on the server, ensuring Whisper, Resemblyzer, and HuggingFace models function without errors.

- **Run End-to-End Tests on Deployed Application**

  Executed the full analysis pipeline online upload, transcription, diarization, sentiment, skills, evaluation, and report export to ensure stable performance across browsers and devices.

Conclusion

The AI Interview Transcript Analyzer represents a comprehensive, intelligent solution that transforms traditional interview evaluation into an automated, data-driven, and highly efficient process. By integrating advanced audio processing, natural language understanding, sentiment modelling, and Gemini-powered reasoning, the system delivers deep insights that would otherwise require extensive manual effort from HR teams. From transcription and speaker diarization to skill extraction and final hiring recommendations, each component works in harmony to provide a complete overview of candidate performance.

Through its modular architecture and interactive Streamlit interface, the platform ensures both technical robustness and exceptional user experience. Recruiters, trainers, and organizations can seamlessly upload audio, explore visually rich insights across dashboards, evaluate candidate strengths, and export professional reports all within a streamlined workflow. The design emphasizes accuracy, transparency, and usability, making it suitable for real recruitment, interview practice, and organizational assessment systems.

Looking forward, the platform provides a strong foundation for enhancements such as multilingual transcription, advanced communication analytics, behavioral scoring, ATS integration, and voice sentiment patterns. With its scalable architecture and expandable AI capabilities, the AI Interview Transcript Analyzer demonstrates how modern AI technologies can revolutionize hiring processes, improve decision-making quality, and set new standards for interview intelligence.