CrackGPT: AI-Powered Interview Simulator

Proejct Report

Abhishek Kumar Jha Aditya Deshmukh Manish Seervi
Mohd Ayaan Subhadeep Sing

1 Project Overview and Motivation

1.1 Problem Statement

Current interview preparation tools have critical gaps:

- Static question banks with no personalization
- No voice-based simulation or soft-skill evaluation
- Limited actionable feedback
- Human mock interviews are expensive and inconsistent

1.2 Solution: CrackGPT

An AI-powered mock interview platform delivering:

- Context-aware question generation from job descriptions and resumes
- Real-time voice interaction with natural conversation flow
- Dynamic follow-ups mimicking human interviewers
- Comprehensive feedback on technical depth and communication
- Live coding environment for DSA challenges

2 Technical Architecture Evolution

2.1 Version 1: Initial Implementation

Technology Stack:

- Whisper (ASR) for speech recognition
- Gemini via AI Studio for LLM inference
- ElevenLabs for text-to-speech
- Streamlit for browser-based audio recording

Critical Limitations: The primary bottleneck was AI Studio's rate limit of only 3 API calls per minute, making real-time conversational flow impossible. Additional issues included:

- No automatic Voice Activity Detection (VAD)
- Manual button-based recording breaking immersion
- Minimal follow-up questions (random generation)
- High latency preventing thorough testing

2.2 Version 2: Google Cloud Migration

To eliminate rate limits, the team migrated to Google Cloud Platform, leveraging $300 free credits.

Enhanced Stack:

- **Google Cloud Speech-to-Text v2:** Real-time transcription with built-in VAD
- **Gemini 2.5 Flash via Vertex AI:** Unlimited rate limits for dynamic conversation
- **Google Cloud Text-to-Speech:** High-quality, low-latency synthesis

- **Flask Backend + Vanilla JS Frontend:** Rapid development and testing

Real-Time Pipeline:

$$\begin{array}{c}\text{User Speech} \rightarrow \text{VAD} \rightarrow \text{Media Recorder} \rightarrow \text{Audio Upload} \\ \downarrow \\ \text{Google STT v2 (1-3s latency)} \rightarrow \text{Transcript} \\ \downarrow \\ \text{Gemini 2.5 Flash} \rightarrow \text{Context-Aware Response} \\ \downarrow \\ \text{Google Cloud TTS} \rightarrow \text{MP3 Data URI} \rightarrow \text{Client Playback}\end{array}$$

Key Improvements:

1. Automatic VAD eliminates manual controls
2. Synchronous STT with 1-3 second latency
3. Unlimited API calls enable multi-turn follow-ups with depth limiting (3-4 per topic)
4. Enhanced audio processing: noise suppression, format transcoding (webm/ogg/opus → WAV PCM)
5. Natural interviewer behavior avoiding robotic phrasing

2.3 ASR vs. STT Analysis

Why ASR Failed:

- Required separate VAD implementation
- Poor pause detection (premature cutoffs or excessive waiting)
- Significant latency with 10-15 second audio chunks
- Synchronization issues between transcript and audio processing

Why Google Cloud STT v2 Succeeded:

- Cloud GPU acceleration (1-3s average latency)
- Integrated VAD with native pause detection
- Handles varying answer durations without manual chunking
- Superior built-in audio enhancement (noise suppression, echo cancellation)

3 MediaPipe Framework

The team initially explored Google's MediaPipe for multimodal analysis (pose estimation, facial landmarks, gesture recognition, eye tracking) to create holistic assessment combining verbal and non-verbal communication.

Key Challenges: Real-time video processing consumed excessive CPU/GPU causing performance degradation; inconsistent browser compatibility (especially Firefox/Safari and mobile); pose estimation struggled with varying lighting and angles; cultural differences in body language made interpretation unreliable; privacy concerns with video/facial tracking.

Strategic Pivot: Abandoned MediaPipe for audio-first approach, gaining universal accessibility (microphone-only), privacy preservation, lower resource usage, and faster development. Retained text-based coding window and transcript display for visual feedback.

4 Implementation Details

4.1 Conversation Management

Sophisticated Prompting System:

1. **Mandatory Question Ending:** Every response must end with "?" (prevents dead-end statements)
2. **Depth Limiting:** Maximum 3-4 consecutive follow-ups per topic, then pivot
3. **Dynamic DSA Integration:** Role-based coding questions (AIML/SWE roles)
4. **Topic Variety:** No repetition within single interview session
5. **Fallback Mechanisms:** Default safe questions when uncertain
6. **Speech Clarity Handling:** Detects spelled-letter transcripts, requests clarification for unclear audio

4.2 Feedback Generation

Structured Output Format:

1. Overview (1-3 sentence summary)
2. Strengths (with rating)
3. Areas for Improvement (concrete bullets $\leq$40 words)
4. Communication & Clarity (rated 1-10 with evidence)
5. Technical Depth & Problem-Solving (grounded in discussion)
6. Actionable Next Steps (3-6 practice recommendations)
7. Suggested Follow-up Questions
8. Overall Rating (holistic score with rationale)

Scoring Guidelines: 9-10 (Outstanding), 7-8 (Strong), 5-6 (Mixed), 3-4 (Weak), 1-2 (Poor)

4.3 Coding Window Implementation

Features:

- Adaptive timer setup with automatic countdown
- Main interview timer pauses during coding session
- Auto-close at timeout; main interview resumes
- Early submission generates reflective follow-up about approach/complexity
- Syntax highlighting for multiple languages

This simulates real whiteboard/coding interview pressure while maintaining conversational flow.

4.4 Integration Architecture

Implementation Journey:

- Initial plan: Separate React.js frontend + Flask backend
- Reality: Major complexity in dynamic audio pipeline
- Solution: Backend-first stabilization with minimal vanilla HTML/CSS/JS for rapid testing
- Gradual UI refinement once backend audio workflows stabilized

Final Architecture:

Frontend (HTML/CSS/JS) $\leftrightarrow$ Flask API (interview_routes, coding_routes)
$\downarrow$
Speech Service (STT/TTS) $\leftrightarrow$ AI Service (Gemini Vertex)
$\downarrow$
Feedback Service (Structured Output)

5 Conclusion

CrackGPT fills key gaps in interview preparation using real-time conversational AI, integrated DSA challenge evaluation, and structured, actionable feedback. The shift from Version 1 (limited by API constraints and MediaPipe complexity) to Version 2 (built on GCP's STT/TTS/LLM stack) shows strong problem-solving and architectural evolution. The platform delivers realistic interview simulations and detailed feedback, helping democratize access to high-quality interview coaching globally.

Team Contributions:

- **Workflow:** Subhadeep Sing, Abhishek Kumar Jha
- **Audio Pipeline:** Aditya Deshmukh, Subhadeep Sing
- **Prompting:** Manish Seervi, Subhadeep Sing
- **Feedback:** Abhishek Kumar Jha, Manish Seervi
- **UI:** Mohd Ayaan, Aditya Deshmukh
- **Integration:** Mohd Ayaan