AI-Powered Excel Mock Interviewer Design Document & Approach Strategy

Deployed Link: https://interview.apkzube.com/

This is a temporary working link and should be expected to function properly within the next 6-10 days, as it has been deployed on AWS EC2.

Complete, Runnable Source Code: GitHub Repository

1. Business Context

- Manual Excel interviews consume senior analyst time and delay hiring.

- Evaluations vary across interviewers, reducing fairness and consistency.

- Hiring pipeline efficiency directly impacts business growth in Finance, Operations, and Data Analytics.

- Solution: An AI-powered system to automate Excel skill assessments while simulating the experience of a real interview.

2. Objectives

- Deliver a web-based AI interviewer that:

* Simulates a structured interview flow.

* Evaluates candidate answers intelligently.

* Maintains conversational state.

* Generates a professional, constructive feedback report.

* Logs transcripts & reports for HR review.

3. Core Requirements

1. Structured Interview Flow – Candidate introduction, multi-turn Q&A, summary.

2. Intelligent Answer Evaluation – LLM-driven evaluation with rubric & fallback keyword checks.

3. Agentic Behavior & State Management – Context-aware, interviewer-like interaction.

4. Constructive Feedback Report – Scores (0–5), rationale, overall rating.

5. Automated Logging – Transcript & feedback auto-saved with timestamp.

4. Technology Stack (Detailed Justification)

1) LLM — Google Gemini (gemini-1.5-flash-latest)
Why this?

- Cost–performance sweet spot: The *Flash* variant is optimized for low latency and lower cost, unlike heavier "Pro" versions. This ensures the system scales affordably during many interview runs.
- Ideal for evaluation workflows: Multi-turn interview flows (Ask → Capture Answer → Evaluate → Feedback) require fast responses without noticeable lag for the candidate.

---

2) Framework — LangChain (Python)
Why this?

- Prompt templates & chaining: Cleanly separates fixed interviewer instructions (role/system prompts) from variable candidate input, making interview flows modular.
- Memory & state management: Built-in conversational memory (buffer/summary) prevents repetitive instructions and helps simulate interviewer "continuity."
- Structured output parsing: With LangChain's parsing tools, you can enforce strict JSON output instead of brittle free-text parsing.
- Flexibility: If later you want to A/B test OpenAI, Anthropic, or open-source LLMs, LangChain provides drop-in integrations.

---

3) Backend — Python Flask
Why this?

- Lightweight & simple: Perfect for serving a REST API that orchestrates LLM calls, handles candidate sessions, and stores logs.
- Production-ready: Combined with Gunicorn + Nginx, Flask apps can be easily containerized and deployed.
- Session management: Flask-Session lets us persist candidate progress across multi-turn interview flows without extra DB overhead.

---

4) Frontend — Flask + Bootstrap 5
Why this?

- Professional, responsive UI: Bootstrap grids, cards, forms, accordions, and progress bars provide a dashboard-like experience without heavy frontend frameworks.
- Fast to iterate: Easy to style Q&A forms, candidate info inputs, and summary views without React/Angular overhead.

- Accessible: Bootstrap provides a11y compliance and responsiveness out of the box, ensuring the interviewer works across desktop and mobile.

---

5) Storage — JSON Question Bank (excel_questions.json)
Why this?

- Human-editable: Hiring managers or Excel SMEs can update/add questions without touching the backend code.
- Version-controllable: JSON lives in Git, so you can track changes over time.
- Zero ops overhead: No database maintenance is required, keeping infra light for MVP.

---

6) Logging — /logs (Transcripts & Feedback Reports)
Why this?

- Traceability: Every interview run produces a transcript + feedback file, which can be used later for audits or training.
- Offline analysis: Recruiters or engineers can mine logs to refine evaluation prompts or spot model failure cases.
- Dataset building: These logs form the cold start dataset for future fine-tuning (Q, candidate answer, LLM eval, human correction).

---

7) Hosting — AWS EC2
Why this?

- Full control: EC2 lets us manage dependencies (Python, system libs) without PaaS constraints.

5. System Architecture

Flow:

1. Candidate enters profile (name, experience).

2. Flask loads questions from excel_questions.json.

3. Candidate answers → LLM evaluates using rubric.

4. JSON {score, rationale} parsed → saved in session.

5. At end → feedback summary computed, transcript + report saved.

6. Interactive summary shown in UI.

6. Approach to Cold Start

A key constraint was the absence of a pre-existing dataset of mock Excel interview transcripts. Our strategy accounts for bootstrapping and improving the system over time:

1. Initial Bootstrapping (Prompt Engineering First):
    - Manually Curated Question Bank: We began by manually creating excel questions.json Each question includes a detailed expected answer description, expected answer keywords, and precise evaluation criteria. This detailed "ground truth" is vital.
    - Intensive Prompt Engineering: In the absence of a training dataset, the LLM's performance hinges on the quality of its prompts. We meticulously designed Prompt Templates for both Excel Evaluator and Feedback Generator, instructing the Gemini LLM to act as an expert, adhere to a strict evaluation rubric, and output structured JSON. This prompt engineering serves as the initial "training" for the AI's specific task.
2. Iterative Improvement Over Time (Data-Driven Refinement):
    - Human-in-the-Loop Feedback: After initial deployments, human Excel experts (e.g., senior analysts) would review the AI's generated evaluations and feedback reports against actual candidate responses. They could provide corrections or adjustments.
    - Prompt Refinement Cycle: This human feedback would directly inform a continuous cycle of prompt refinement. If the AI consistently misunderstands a nuance or misjudges an answer, the prompts (especially evaluation template and feedback template) would be updated to provide clearer instructions, more examples, or specific negative constraints.
    - Building a Dataset: The automated logging of full interview transcripts (transcript_*.txt) and detailed evaluations stored within interview history forms the foundation for building a proprietary dataset.
        - These collected Q&A pairs, coupled with the AI's evaluation and *human-corrected/validated* scores and justifications, would gradually create a valuable, domain-specific dataset.
    - Future Model Optimization (Fine-tuning): Once a sufficient human-validated dataset is accumulated, it could be used for:
        - Fine-tuning smaller LLMs: Training a more specialized, potentially more efficient and cost-effective LLM (e.g., an open-source model or a custom Gemini model) specifically on Excel interview evaluations. This would reduce reliance on general-purpose models for every call.
        - Developing dedicated evaluation models: For high-volume scenarios, a separate, smaller machine learning model could be trained to

predict scores or classify answer quality, further optimizing performance and cost.

- A/B Testing: Future iterations could involve A/B testing different prompt versions or model configurations to quantitatively measure improvements in evaluation accuracy and candidate experience.

7.Future Optimization:

- ✓ Fine-tune smaller open-source LLMs.
- ✓ Skill-wise analytics dashboard.
- ✓ Adaptive questioning (difficulty adjustment).
- ✓ ATS integration.

8.Challenges & Learnings

- LLM Quotas: Gemini free-tier (50 requests/day) exhausted quickly → added fallback keyword evaluation.
- Output Parsing: LLM sometimes returned invalid JSON → built regex-based JSON extraction + error handling.
- Cold Start: No dataset available → bootstrapped with curated JSON question bank.
- LangChain Errors: Conversation setup gave errors → fixed by adjusting how prompts were added.