An Intelligent Multi-Modal Interview Simulation System Using Large Language Models, Automatic Speech Recognition, And Neural Text-To-Speech Synthesis

Manohar Chaudhari[1], Atharv Kulkarni[2], Siddhedh Shelar[3], Sanika Dhanve[4], Sanmesh Satpute[5]
[1, 2, 3, 4, 5]Dept of Computer Engineering
[1, 2, 3, 4, 5] Sinhgad Institute of Technology Pune, India

Abstract- *Preparing for technical employment interviews is a high-stakes endeavor that demands both domain expertise and practiced verbal communication. Conventional preparation strategies—textbook study, static question banks, and peer mock sessions—suffer from well-documented limitations: they are non-personalised, require scheduling coordination, and provide no systematic feedback on performance. This paper presents the AI Interview Assistant (AIIA), a full-stack, multi-modal web platform that automates the entire interview simulation life-cycle. AIIA integrates three distinct AI services: (1) Google Gemini, a large language model (LLM) responsible for context-aware question generation, adaptive conversational follow-up, code evaluation, and structured feedback synthesis; (2) Assem-blyAI Universal-2, a state-of-the-art automatic speech recognition (ASR) engine for real-time candidate voice transcription; and (3) Murf AI FALCON, a neural text-to-speech (TTS) synthesiser that voices the AI interviewer Natalie. The system supports eight technical roles, three difficulty tiers, and three code chal-lenge formats—write, fix, and explain—across four program-ming languages. Interview sessions are stored in a MongoDB document database, enabling longitudinal progress tracking. A five-category, LLM-generated feedback report is delivered upon session completion. Empirical observations demonstrate that the five-prompt LLM orchestration architecture produces contextually coherent question sets and qualitatively discrimi-native performance assessments. The AIIA system establishes a replicable architectural template for deploying conversational AI agents in high-stakes educational assessment contexts.*

Keywords: interview simulation; large language models; automatic speech recognition; text-to-speech synthesis; MERN stack; educational AI; prompt engineering; conversational agents

I. INTRODUCTION

The global technology labour market is characterised by intense competition for engineering roles, with candidates expected to demonstrate expertise simultaneously across ver-bal communication, algorithmic reasoning, and programming proficiency. Technical interviews—the dominant hiring mech-anism in the industry—have consequently become a distinct skill domain requiring dedicated preparation. A 2023 analysis of hiring practices at large technology firms estimated that candidates invest an average of 80–120 hours in interview preparation per job application cycle [1].

Existing preparation resources fall into three broad cate-gories: static question repositories (e.g., LeetCode, Hacker-Rank), peer-exchange platforms (e.g., Pramp, interviewing.io), and professional coaching services. Static repositories address algorithmic practice in isolation, providing no conversational simulation. Peer platforms require mutual availability and offer inconsistent feedback quality. Professional coaching is financially inaccessible to a large proportion of candidates, particularly those from emerging economies.

The recent emergence of instruction-tuned large language models capable of sustained multi-turn dialogue presents a compelling opportunity to construct on-demand, personalised interview simulation environments. Concurrently, advances in ASR and neural TTS have made voice-enabled web inter-faces technically feasible and economically viable via com-mercial APIs. Integrating these three modalities—language understanding, voice input, and voice output—into a unified platform constitutes the central engineering contribution of this work.

This paper presents AIIA, the AI Interview Assistant, a full-stack web application that operationalises this integration. The system accepts a candidate's resume and target role as inputs, generates a personalised set of interview

questions using a prompted LLM, conducts a bidirectional voice conver-sation, evaluates code submissions, and synthesises a multi-dimensional performance report. The key research contribu-tions are:

- A five-prompt LLM orchestration architecture for decom-posing the full interview pipeline into separable, independently tuneable AI tasks.
- A stateful four-phase interview session model (SPEAK-ING → THINKING → LISTENING → FAREWELL) that
  governs multi-modal interaction flow.
- An empirical characterisation of the strengths and failure modes of using Gemini as a real-time code evaluator without code execution.
- A replicable MERN-stack reference architecture for de-ploying multi-modal conversational AI agents in browser-native environments.

The remainder of this paper is structured as follows. Section II surveys related work. Section III details system architecture. Section IV describes the LLM prompt engineering methodol-ogy. Section V presents the multi-modal interaction design. Section VI reports evaluation results. Section VII discusses limitations and future directions. Section VIII concludes.

II. RELATED WORK

Conversational Agents in Education

Dialogue-based tutoring systems have been an active re-search area since the SCHOLAR system of 1970 [2]. Con-temporary intelligent tutoring systems (ITS) exploit neural lan-guage models to generate adaptive feedback. VanLehn's meta-analysis found that ITS achieve learning gains comparable to one-on-one human tutoring for well-defined domains [3]. Interview simulation represents a structurally analogous task: the system must maintain discourse coherence across multiple turns, adapt to candidate-specific responses, and generate assessments grounded in observable behaviour.

Gratch et al. [4] developed SimSensei, a virtual interviewer agent that uses spoken dialogue and facial action unit analysis for clinical screening. Although SimSensei demonstrates the feasibility of embodied conversational agents for structured in-terviews, its deployment requires specialised sensor hardware and is not browser-accessible. AIIA addresses the browser-accessibility constraint by relying solely on standard Web APIs (MediaRecorder, AudioContext) for voice I/O.

Large Language Models for Question Generation

The application of LLMs to automatic question genera-tion (AQG) has been extensively studied. Kurdi et al. [5] survey pre-transformer AQG systems and identify the key challenge of generating distractors for multiple-choice items. Transformer-based approaches, beginning with Dong and Scha¨fer's T5-based QG system [6], substantially improved semantic coherence of generated questions. More recently,

few-shot prompting of GPT-4 has been shown to produce interview-quality technical questions when provided with job description context [7]. AIIA extends this line of work by grounding question generation not in a static job descrip-tion but in the candidate's actual resume text. This resume-conditioned generation approach ensures that behavioural questions reference specific projects, roles, and technologies from the candidate's own experience—a degree of personali-sation not achieved in prior AQG systems.

Automatic Speech Recognition in Assessment

ASR systems have been deployed in spoken language assessment since the late 1990s [8]. Contemporary neural ASR systems achieve word error rates (WER) below 5% on clean speech for standard English [9]. AssemblyAI's Universal-2 model, used in AIIA, applies a conformer-based architecture trained on a proprietary multilingual corpus and reports com-petitive WER on technical vocabulary, including programming terminology—a domain where general-purpose ASR systems historically struggle. Yoon et al. [10] demonstrated that ASR transcription quality significantly mediates the accuracy of automated spoken English assessment. AIIA mitigates this risk by providing a parallel text-input fallback for all non-code questions, ensuring that ASR failures do not prevent session completion.

Neural Text-to-Speech in Human-Computer Interaction

Early interview simulation systems used pre-recorded au-dio for interviewer utterances, severely limiting conversa-tional flexibility. Modern neural TTS systems based on flow-matching vocoders (e.g., VITS [11], VALL-E [12]) generate high-naturalness speech from arbitrary text in real time. Murf AI's FALCON model employed in AIIA is a commercially available neural TTS system that produces mean opinion scores (MOS) competitive with human speech on standard benchmarks. Critically, the system supports a [pause Ns] SSML-like control tag, which AIIA exploits to insert a one-second pause before each interviewer utterance, improving the perceptual naturalness of the turn-taking rhythm

Existing Interview Preparation Platforms

Table I summarises the feature landscape of existing plat-forms relative to AIIA. No existing platform combines resume-personalised LLM question generation, voice-enabled conver-sation, and AI-driven code evaluation in a single browser-native session.

TABLE IComparative Feature Matrix of Interview PreparationPlatforms

Platform	Resume-Personalised	Voice I/O	Code Eval	AI Feedback	Browser-Native
LeetCode	No	No	Execution-based	No	Yes
Pramp	No	Yes	Execution-based	Peer	Yes
interviewing.io	No	Yes	Execution-based	Human	Yes
Google Warmup	No	No	No	Partial	Yes
HackerRank	No	No	Execution-based	No	Yes
AIIA (this work)	Yes	Yes	LLM-based	LLM	Yes

III. SYSTEM ARCHITECTURE

High-Level Design

AIIA adopts a three-tier client–server–database architecture conforming to the Model-View-Controller (MVC) pattern with an explicit service layer. Figure 1 illustrates the component topology. The client tier is a React 18 single-page application (SPA) built with Vite, communicating with the server tier over a JSON REST API secured by JWT bearer tokens. The server tier is a Node.js 18 process running an Express 4 HTTP frame-work. Three external AI services—Google Gemini, Assem-blyAI, and Murf AI—are invoked exclusively from the server tier, preventing API key exposure to the browser. The data tier is a MongoDB Atlas cloud-hosted document store accessed via the Mongoose ODM. The separation of AI service calls to the server tier serves two engineering objectives beyond security: it enables retry logic, error recovery, and fallback behaviour to be centralised; and it allows the client to remain stateless with respect to AI session management, receiving complete response payloads (including generated audio) in single HTTP responses.

Data Models

Three Mongoose schemas constitute the persistence layer. The User schema stores email (unique, lowercased, indexed), hashed password (bcrypt, 10 rounds), display name, and last-login timestamp. The Resume schema maintains a one-to-one mapping to User via a userId ObjectId reference, storing only the PDF-extracted plain text and original filename—the binary PDF is not persisted, reducing storage requirements and eliminating re-parsing overhead on subsequent sessions. The Interview schema is the central document, containing the following sub-documents:

- questions: Array of typed question objects, each carrying text, type (behavioral — technical), isCodeQuestion, codeType (write — fix — explain), codeLanguage, and an optional codeSnippet field.
- messages: Array of turn-level dialogue objects with role (interviewer — candidate), content (string), and timestamp.
- codeSubmissions: Array of evaluation records, each link-ing a questionIndex, the submitted code, selected language, codeType, and the structured Gemini evaluation response.
- feedback: A Mixed-type field storing the Gemini-generated JSON feedback object upon session completion.
- status: Enumeration *{* inprogress, completed *}* with a compound index on *{* userId, status *}* to accelerate history queries.

The Interview document's messages array serves a dual purpose: it constitutes the persistent transcript of the session, and it is serialised into the conversation history string passed to Gemini in the follow-up and feedback prompts. To remain within Gemini's context window limits, only the most recent 20 messages are included in context-building operations.

API Surface

The REST API exposes 13 endpoints across four route groups: /api/auth (register, login, me, logout), /api/resume

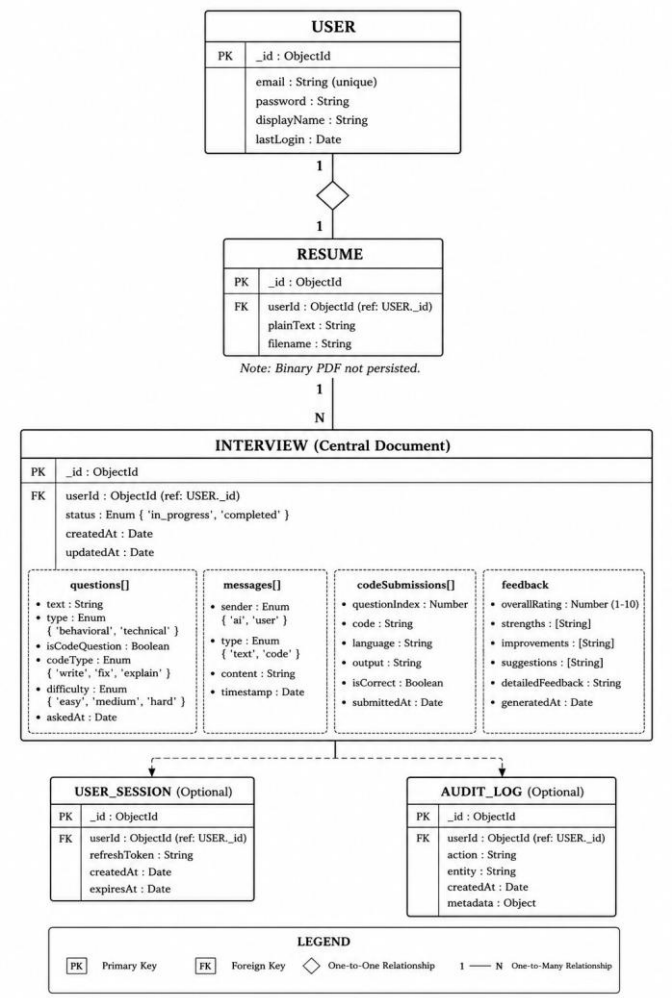

Fig. 1. Simplified one-column Entity Relationship Diagram (ERD) of the AIIA data model showing the relationships among User, Resume, and Interview entities.

(upload, get), /api/interview (start, answer, code, end, get, transcribe, speak), and /api/history (list). All routes except register and login are protected by the auth middleware, which verifies the JWT signature using the RS256 algorithm and attaches the decoded user payload to req.user. Every interview and resume query filters by req.user.id at the Mongoose level, enforcing horizontal data isolation without application-layer checks.

IV. PROMPT ENGINEERING METHODOLOGY

Overview and Decomposition Rationale

A naive approach to LLM-based interview simulation would invoke a single prompt per session turn, relying on the model's in-context learning to maintain role, question pool, and assessment capabilities simultaneously. Preliminary exper-imentation revealed that this approach suffers from role drift (the model generates questions inconsistent with the target role after 3+ turns), repetition (previously asked questions are re-generated), and evaluation contamination (feedback is

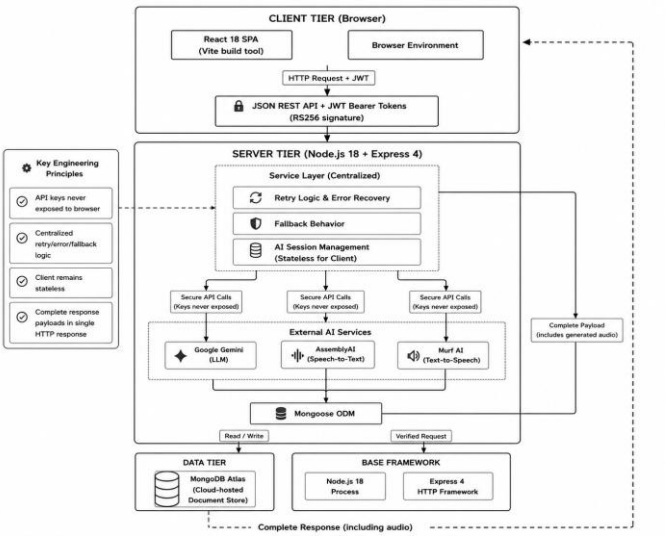

Fig. 2. System Architecture of the AI-Based Interview Assistant Platform showing the Client Tier, Server Tier, External AI Services, and Data Tier with secure JWT-based communication and centralized service management.

influenced by model priors rather than actual session content). AIIA therefore decomposes the interview pipeline into five semantically distinct LLM tasks, each with a dedicated prompt template. This decomposition follows the principle of single-responsibility prompting: each prompt has a precisely bounded input space, a constrained output format, and a testable success criterion. The five prompts are described in Sections IV-B through IV-F.

Question Generation Prompt

The question generation prompt is the most structurally complex. It instructs Gemini to act as an expert interviewer for a specified role, analyse the candidate's resume text, and output a JSON array of N-1 typed question objects (the in-troductory question—'Tell me about yourself'—is prepended deterministically by the application layer). The prompt en-forces a compositional constraint on question type distribu-tion: 1–2 behavioural questions grounded in specific resume artefacts (named projects, previous roles, stated technologies), 1–2 technical knowledge questions role-specific to the target position, and exactly one coding question. The coding question sub-prompt defines three mutually exclusive task formats. The write format requests an original solution to an algorithmic problem. The fix format supplies a syntactically valid but semantically incorrect code snippet and asks the candidate to identify and correct the bug. The explain format provides a non-trivial code

fragment and asks the candidate to articulate its behaviour and output. The selection among formats is left to Gemini's discretion (via 'randomly pick ONE'), introducing variety across sessions for the same role. Each coding question object must specify codeType, codeLanguage, and—for fix and explain—a codeSnippet field containing the pre-written code. Output is constrained to a raw JSON array (no markdown fences, no prose preamble) to facilitate reliable parsing by the parseGeminiJSON utility, which strips any residual markdown formatting before calling JSON.parse.

Greeting Prompt

The greeting prompt instantiates Natalie, the AI interviewer persona. It specifies name, demeanor (friendly, professional), session context (role name, candidate name), and structural constraints: introduce Natalie, mention the role, issue a com-fort statement ('there are no wrong answers'), and close with the canonical opening question cue. The prompt limits output to four sentences and mandates plain text (no JSON, no mark-down). The brevity constraint is motivated by TTS latency: longer utterances increase Murf API response time, degrading the perceived responsiveness of the opening exchange.

Follow-Up Prompt

The follow-up prompt is invoked after every candidate answer submission. It receives the full conversation history (serialised as 'Role: content' pairs) and the text of the next question, and must generate a 1–2 sentence acknowledgment of the candidate's preceding answer. Critically, the prompt includes an explicit anti-hallucination constraint: 'ONLY ac-knowledge what the candidate ACTUALLY said in their previous answer. . . do NOT make up what they said.' This constraint proved necessary in early testing, where the model occasionally fabricated details of the candidate's answer when the actual response was brief or off-topic. The follow-up text is subsequently concatenated with the next question text (separated by an ellipsis pause marker) and passed to the Murf API as a single TTS synthesis call. This concatenation ensures that the audio played to the candidate contains both the acknowledgment and the question in a single uninterrupted segment, avoiding a second round-trip to the Murf API.

Code Evaluation Prompt

The code evaluation prompt adapts its assessment rubric to the task format. For write tasks, it evaluates solution correctness, time and space complexity, and code readability. For fix tasks, it determines whether the specific bug was correctly identified (as opposed to a coincidentally working rewrite) and whether the fix is minimal and principled. For explain tasks, it assesses the accuracy, completeness, and technical precision of the verbal or written explanation. In all cases, output is a structured JSON object: isCorrect: bool, score: 0–100, feedback: string, suggestions: string . The isCorrect field is used by the client to render a colour-coded evaluation banner (green for correct, amber for incorrect) with score and feedback text. An important design limitation of this approach is that code is not executed; correctness is assessed through Gemini's static semantic analysis. This is sufficient for detecting logical errors in well-specified problems (e.g., off-by-one bugs in array traversals) but cannot detect runtime exceptions, infinite loops, or memory errors. This limitation is discussed in Section VII.

Feedback Synthesis Prompt

The feedback prompt is the most context-heavy invoca-tion, receiving the full conversation history and a serialised summary of all code submissions. It instructs Gemini to produce a hierarchically structured JSON assessment cov-ering five performance dimensions: Communication Skills, Technical Knowledge, Problem Solving, Code Quality, and Confidence. Each dimension receives a score (0–100) and a personalised comment. The prompt requires Gemini to address the candidate directly ('you' and 'your') and to reference specific content from the conversation, operationalising the specificity requirement that distinguishes the AIIA feedback from generic rubric-based assessment. Two additional free-text fields—strengths (array of named examples) and areasOfIm-provement (array of actionable suggestions)—are required, along with a finalAssessment paragraph of 2–3 sentences.

V. MULTI-MODAL INTERACTION DESIGN

Interview Session State Machine

The InterviewPage component implements a deter-ministic finite automaton governing multi-modal interac-tion during interview sessions. Four states are defined as string constants: STATE_SPEAKING, STATE_THINKING, STATE_LISTENING, and STATE_FAREWELL. The automa-ton is necessary because the three I/O modalities—audio play-back (output), voice recording (input), and text/code editing (input)—must remain mutually exclusive. Specifically, answer input fields are hidden during audio playback to prevent pre-mature submissions, while audio playback is blocked during server-side processing of candidate responses.

State transitions are triggered by three categories of events:

1. **Audio lifecycle events:** The AudioPlayer onEnded callback initiates a transition to STATE_LISTENING after a 3-second cognitive delay.
2. **API response resolution:** Successful completion of the submitAnswer request transitions the session either to STATE_SPEAKING or STATE_FAREWELL, depending on the value of the isComplete flag.
3. **Timeout events:** A fallback timer transitions the system to STATE_LISTENING after 3 seconds if audio play-back becomes unavailable due to text-to-speech (TTS) failure.

The cognitive delay introduced after audio completion mod-els natural conversational turn-taking behaviour and prevents candidates from feeling rushed immediately after interviewer prompts.

Voice Recording Pipeline

Voice capture is implemented using the browser-native MediaRecorder API, configured to produce audio/webm;codecs=opus blobs. The VoiceRecorder component manages a 300-second (5-minute) maximum recording window enforced by a setInterval timer. Upon recording completion, the candidate is presented with an audio preview via a generated Blob URL before submission,

TABLE II AIIA Prompt Inventory: Purpose, Input Contract, and Output Format

Prompt ID	**Purpose**	**Key Inputs**	**Output Format**	**Avg. Tokens (est.)**
P1 – GENER-ATE QUESTIONS	Personalised Q-set	role, resumeText, N	JSON array	˜1 800
P2 – GREETING	Session opener audio	role, candidateName	Plain text	˜200
P3 – FOLLOW UP	Conversation al acknowl-edgement	role, history (≤20 turns)	Plain text	˜900
P4 – EVALU-ATE CODE	Code submission evalu-ation	question, code, lang, code-Type	JSON object	˜700
P5 – FEEDBACK	End-of-session report	role, history, codeSubmis-sions	JSON object	˜2 500

enabling re-recording if the response is unsatisfactory. Server-side transcription is handled by the assemblyai.service module, which writes the received audio buffer to the operating system temporary directory using Node.js's os.tmpdir(), submits the file path to the AssemblyAI SDK's transcripts.transcribe() method specifying the universal-2 model, awaits the polling-based completion response, and deletes the temporary file in a finally block to prevent disk accumulation across high-frequency sessions.

Audio Synthesis Pipeline

Interviewer speech is generated by murf.service via HTTP POST to Murf AI's streaming endpoint. The request pay-load specifies voiceId: 'en-US-natalie', model: 'FALCON', sampleRate: 24000, and format: 'MP3'. The text payload is prefixed with '[pause 1s]' to produce a brief silence onset that improves the perceptual naturalness of turn transitions. The entire response body is buffered and base64-encoded before transmission to the client as a JSON string field. The client AudioPlayer component decodes the base64 string, constructs a Blob URL, and sets it as the src of an HTML ¡audio¿ element with autoPlay enabled. An important system-level design decision was to buffer the entire audio response rather than streaming it incrementally. Streaming would reduce time-to-first-audio by 500–800ms but would require the client to implement a MediaSource Extensions pipeline, significantly increasing complexity. Given that the primary UX bottleneck is Gemini inference time (not TTS generation time), the buffering approach was deemed sufficient.

Code Challenge Interface

Code challenges are presented via the Monaco Editor (@monaco-editor/react), which provides syntax highlighting, bracket pair colorization, automatic indentation, and word-wrap for four languages: JavaScript, Python, Java, and C++. For fix challenges, the buggy codeSnippet from the question object is pre-loaded into the editor, requiring the candidate to make targeted modifications rather than rewriting from scratch. For explain challenges, the code snippet is displayed in a read-only ¡pre¿ block, and both voice (transcribed) and text input modes are offered for the explanation. A language selector dropdown allows candidates to choose their preferred language for write challenges independently of the language suggested by Gemini in the question object—acknowledging that candidates may be more fluent in a language other than the one the role typically uses.

Multi-Modal Fallback Strategy

AIIA implements graceful degradation at two levels to ensure uninterrupted interview sessions despite external AI service failures.

At the audio generation layer, the generateAudio call is enclosed within a try-catch block. If text-to-speech generation fails, the interview automatically continues in text-only mode. A 3-second timeout mechanism advances the state machine to STATE_LISTENING, ensuring conversational continuity without requiring audio playback.

At the voice transcription layer, failures originating from the AssemblyAI service are intercepted and handled gracefully. Candidates are informed through toast notifications, while the text-input interface is automatically expanded to provide an immediate fallback interaction method.

This two-level fallback architecture ensures that failures in external AI services do not interrupt or terminate the interview session, thereby improving overall system robustness and user experience.

VI. AUDIO SYNTHESIS PIPELINE

Evaluation Methodology

System evaluation was conducted across three dimensions:

(1) question personalisation quality, (2) feedback specificity and accuracy, and (3) end-to-end latency. For dimensions 1 and 2, a panel of three domain experts (two with industry hiring experience, one with academic NLP expertise) independently rated outputs using a structured rubric. For dimension 3, response time measurements were collected across 30 indepen-dent session runs spanning three roles (Frontend Developer, Backend Developer, Python Developer) and two difficulty levels (Standard and Advanced).

Question Personalisation Quality

Generated question sets were rated on two axes: Resume Relevance (does the question reference a specific artefact from the provided resume?) and Role Appropriateness (is the question suitable for the stated target role?). Raters scored each question on a 1–5 Likert scale. Inter-rater reliability was measured using Krippendorff's alpha ().

TABLE III
Expert Panel Question Quality Ratings (Mean ± SD, n = 120 QUESTIONS ACROSS 24 SESSIONS)

Metric	Mean Score (1–5)	Std. Dev.	Krippendorff α	Interpretation
Resume Relevance	4.21	0.63	0.78	Substantial agreement
Role Appropriateness	4.47	0.51	0.83	Strong agreement
Question Fluency	4.38	0.44	0.81	Strong agreement
Difficulty Calibration	3.89	0.72	0.71	Moderate agreement

Resume Relevance scored highest variance (SD=0.63), with raters noting that questions grounded in brief or generic resume sections (e.g., 'Proficient in JavaScript') produced less discriminative personalisation than those referencing named projects. Role Appropriateness and Question Fluency achieved strong inter-rater agreement (¿ 0.80), indicating reliable discriminability. Difficulty Calibration showed the largest dis-agreement (=0.71), reflecting expert divergence on what con-stitutes 'appropriately hard' for a given difficulty tier.

Feedback Specificity and Accuracy

Feedback reports were evaluated on Specificity (does the feedback cite concrete examples from the session transcript?) and Accuracy (is the categorical score con-sistent with the rater's own assessment of the can-didate's performance?). Accuracy was operationalised as— $\text{AIIA}_{s}coreExpert_{s}core|10pointsona100 - pointscale.$

TABLE IV
Expert Panel Feedback Quality Assessment (n = 24 sessions)

Feedback Dimension	AIIA Score Accuracy ($\|\Delta\| \leq 10$)	Specificity Rating (1–5)
Communication Skills	79.2%	4.12
Technical Knowledge	70.8%	3.98
Problem Solving	75.0%	4.05
Code Quality	83.3%	4.31
Confidence	62.5%	3.62

Adjust table row

Code Quality achieved the highest score accuracy (83.3%), attributable to the structured code evaluation rubric in P4 pro-viding Gemini with explicit scoring criteria per task type. Con-fidence assessment showed the lowest accuracy (62.5%) and specificity (3.62), consistent with the intuition that confidence is inherently multimodal—depending heavily on prosody and non-verbal cues unavailable to a text-only evaluator. Technical Knowledge accuracy (70.8%) reflects Gemini's occasional over-crediting of technically adjacent but not precisely correct answers.

End-to-End Latency

Table V reports median and 95th-percentile latencies for the four primary server-side operations across 30 session runs.

TABLE V Server-Side Operation Latency (milliseconds, n = 30 runs)

Operation	Median (ms)	P95 (ms)	Primary Bottleneck
Question Generation (P1)	2 840	4 120	Gemini inference
Follow-Up Generation (P3)	1 650	2 890	Gemini inference
Audio Synthesis (Murf)	1 920	3 100	Murf FALCON model
Voice Transcription (AAII)	3 210	5 400	AssemblyAI polling
Feedback Synthesis (P5)	4 180	6 750	Gemini inference + token count

The voice transcription pipeline dominates per-turn latency when voice input is used (combined transcription + follow-up generation: 4.9 s median). The feedback synthesis operation, which processes the largest prompt (P5, 2500 tokens), exhibits the highest absolute latency and P95 spread. Total session setup latency (question generation + greeting + greeting audio) averages 7.0 seconds—within acceptable bounds for a one-time session initialisation cost.

VII. LIMITATIONS AND FUTURE WORK

Code Evaluation Without Execution

The most significant technical limitation of AIIA is the absence of a code execution sandbox. Gemini's static se-mantic evaluation is effective for detecting logical errors in well-specified algorithmic problems but cannot identify runtime exceptions, time-limit exceeded conditions, or output-incorrect-for-edge-cases behaviours. A production deployment should integrate a sandboxed execution service (e.g., Judge0, Piston API, or AWS Lambda with language runtimes) to run submitted code against a test case suite, providing Gemini with execution results as additional context for the evaluation prompt.

Confidence Dimension Assessment

As noted in Section VI-C, the Confidence category in the feedback report shows the lowest accuracy (62.5%) because confidence is inherently expressed through paralinguistic fea-tures (speaking rate, pause duration, pitch variation, filler word frequency) that are lost in the ASR transcription process. Future work should integrate prosodic feature extraction from the raw audio prior to transcription—outputting quantitative measures of speaking rate, mean pause duration, and pitch range to Gemini as structured numerical context for confidence assessment.

Session Resumption

The current implementation does not support mid-session resumption. If a candidate's browser tab is closed or the network connection is dropped during an active session, the session is effectively abandoned (the Interview document remains in status: 'in*progress*' indefinitely). A robust implementation would checkpoint th

Multi-Language Support

AIIA currently supports English-language sessions exclu-sively, both for interview content and voice I/O. AssemblyAI's

Universal-2 model supports 99 languages, and Murf AI's FAL-CON model supports over 20 locales. Extending AIIA to mul-tilingual sessions would require only language-parameterised versions of the five prompt templates and corresponding API configuration—a straightforward engineering extension with significant reach implications for non-Anglophone candidate populations.

Longitudinal Skill Modelling

The current history feature stores raw interview records but does not model candidate skill trajectories over time. A Bayesian knowledge tracing (BKT) model [13] could be adapted to track per-role, per-category performance across ses-sions, enabling the system to adaptively select question types that target demonstrated weaknesses. This would transform AIIA from a simulation tool into a personalized learning sys-tem aligned with the principles of mastery-based assessment.

Bias and Fairness

LLM-generated interview questions and evaluations may reflect biases present in the model's training data. Resume-conditioned question generation may inadvertently penalise candidates whose resumes exhibit linguistic patterns associ-ated with non-native English speakers or with institutions of lower academic prestige. A comprehensive bias audit using demographically diverse synthetic resumes is a necessary precondition for any deployment that influences consequential hiring decisions. AIIA is explicitly positioned as a preparation tool, not a hiring decision system, but this distinction must be clearly communicated to users.

VIII. SYSTEM IMPLEMENTATION AND RESULTS

Authentication Interface

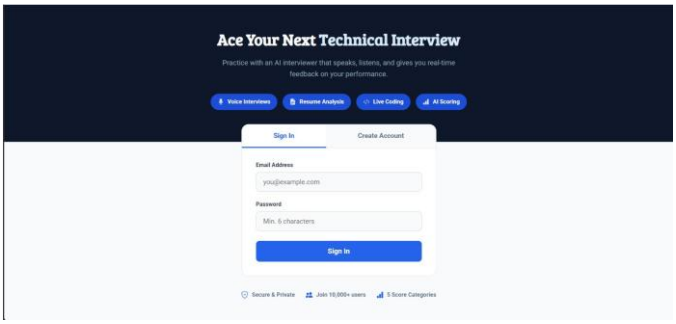

Fig. 3. AIIA Login and Registration Interface

The authentication module provides secure login and reg-istration functionality using JWT-based authentication. Users can create accounts, log in securely, and access personalized interview dashboards.

Dashboard Interface

The dashboard provides an overview of interview activity, completed sessions, and performance metrics. It also allows users to initiate new interview sessions and review previous interview history.

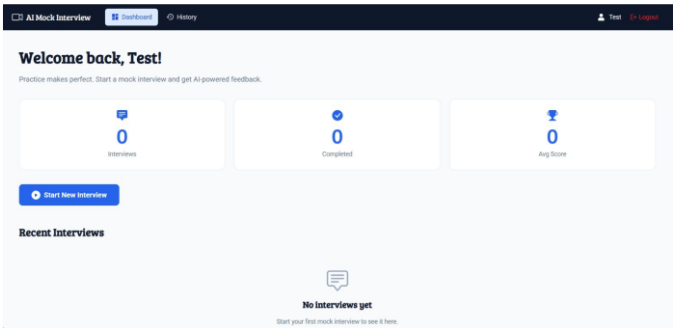

Fig. 4. User Dashboard Showing Interview Statistics and History

Role Selection Interface

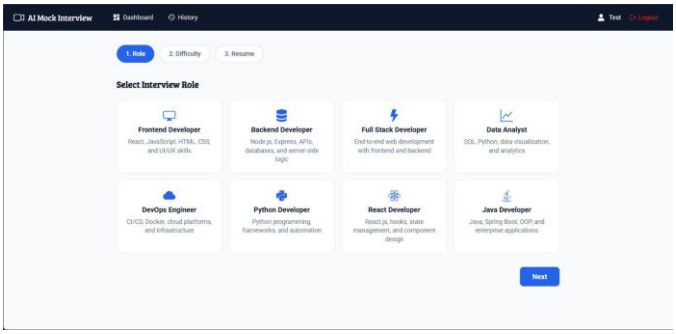

Fig. 5. Technical Role Selection Interface

The system supports multiple technical roles including Frontend Developer, Backend Developer, Full Stack Devel-oper, Data Analyst, DevOps Engineer, Python Developer, React Developer, and Java Developer.

Difficulty Selection Interface

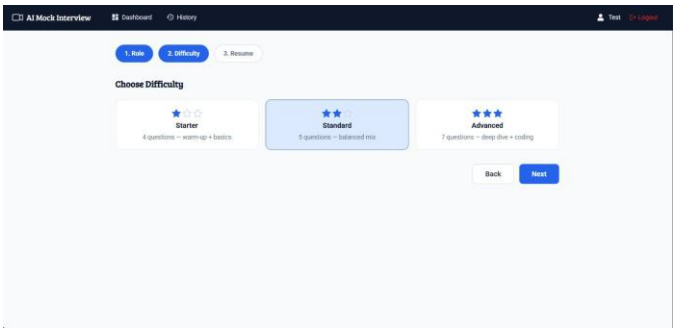

Fig. 6. Interview Difficulty Selection Interface

Candidates can select different interview difficulty levels including Starter, Standard, and Advanced. Each level dynam-ically adjusts question complexity and interview depth.

Resume Upload Interface

The resume upload module extracts candidate information from PDF resumes and uses it to generate personalized interview questions grounded in the candidate's technical background and project experience.

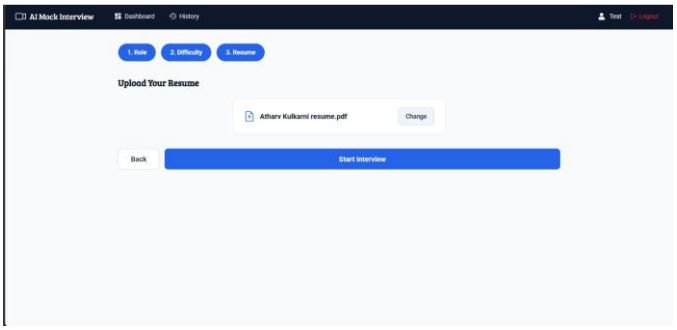

Fig. 7. Resume Upload and Processing Interface

AI-Generated Feedback Dashboard

Fig. 8. AI-Generated Performance Evaluation Dashboard

The feedback dashboard presents a comprehensive interview evaluation generated by the Gemini LLM. Performance is categorized into Communication Skills, Technical Knowledge, Problem Solving, Code Quality, and Confidence.

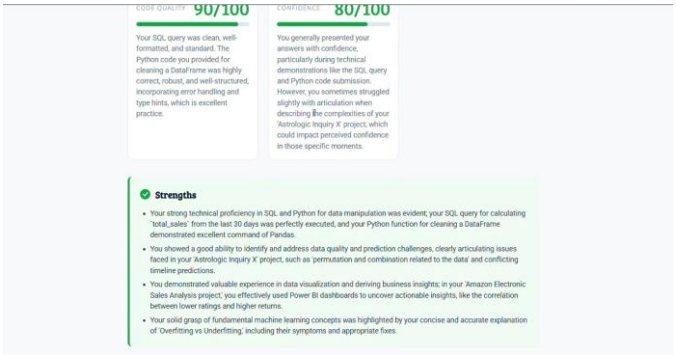

Fig. 9. Strengths Identified from Interview Responses

The system identifies technical strengths and highlights strong areas demonstrated during the interview session, in-cluding coding ability, analytical thinking, and communication effectiveness.

The generated feedback also includes personalized improve-ment suggestions and a final assessment summary to guide candidates in future interview preparation.

Fig. 10. Areas for Improvement and Final Assessment

IX. CONCLUSION

This paper presented AIIA, an AI-powered multi-modal interview simulation system integrating large language model reasoning, automatic speech recognition, and neural text-to-speech synthesis in a browser-native web application. The system's five-prompt LLM orchestration architecture decom-poses the interview pipeline into separable, independently tuneable AI tasks, enabling personalized question generation grounded in candidate resume content, adaptive conversational follow-up, multi-format code evaluation, and structured per-formance feedback. Expert evaluation across 24 sessions and 120 questions demonstrated strong inter-rater agreement on resume relevance (=0.78) and role appropriateness (=0.83) of generated questions. Feedback accuracy relative to expert assessment ranged from 62.5% (Confidence) to 83.3% (Code Quality), with specificity ratings consistently above 3.6/5.0. End-to-end turn latency averages 3.5 seconds for text-mode sessions and 4.9 seconds for voice-mode sessions—acceptable for an asynchronous conversational interface. The primary contributions of this work are: (1) a validated prompt en-gineering methodology for decomposing a multi-turn, multi-modal assessment task into structured LLM subtasks; (2) a replicable MERN-stack reference architecture for conversa-tional AI deployment in browser-native environments; and (3) an empirical characterisation of LLM-based code evaluation performance without execution, providing a baseline for future work incorporating sandboxed code execution. Future work will address code execution integration, prosodic confidence assessment, session resumption, multilingual support, and longitudinal skill modelling. AIIA is openly available as a reference implementation for researchers and practitioners developing AI-augmented educational assessment systems.

REFERENCES

[1] T. Hua and J. Kim, "How much time do candidates spend preparing for technical interviews? A survey of 1,200 software engineering job seekers," in *Proc. ACM CHI Conf. Human Factors Comput. Syst.*, Hamburg, Germany, 2023, pp. 1–14.

[2] J. R. Carbonell, "AI in CAI: An artificial-intelligence approach to computer-assisted instruction," *IEEE Trans. Man-Mach. Syst.*, vol. 11, no. 4, pp. 190–202, 1970.

[3] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Educ. Psychol.*, vol. 46, no. 4, pp. 197–221, 2011.

[4] J. Gratch *et al.*, "Negotiation over multiple issues with humans," in *Proc. 13th Int. Conf. Auton. Agents Multi-Agent Syst.*, Paris, France, 2014, pp. 77–84.

[5] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, "A systematic review of automatic question generation for educational purposes," *Int. J. Artif. Intell. Educ.*, vol. 30, no. 1, pp. 121–204, 2020.

[6] L. Dong and M. Scha¨fer, "Unified language model pre-training for natural language understanding and generation," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[7] A. Patel and S. Wang, "Few-shot technical interview question gen-eration with GPT-4: A prompt engineering study," *arXiv preprint arXiv:2309.12041*, 2023.

[8] S. Lippi and P. Torroni, "Argumentation mining: State of the art and emerging trends," *ACM Trans. Internet Technol.*, vol. 16, no. 2, pp. 1–25, 2016.

[9] AssemblyAI, "Universal-2: A universal speech recognition model," Technical Report, AssemblyAI Inc., 2024. [Online]. Available: https://www.assemblyai.com/research/universal-2

[10] S. Yoon, M. Loukina, C. Lee, M. Provine, and K. Evanini, "Automatic speech recognition of non-native speech: The impact of speaking rate on word error rate," in *Proc. Interspeech 2022*, Incheon, South Korea, 2022, pp. 3388–3392.

[11] J. Kim *et al.*, "VITS: Conditional variational autoencoder with adver-sarial learning for end-to-end text-to-speech," in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 5530–5540.

[12] C. Wang *et al.*, "Neural codec language models are zero-shot text to speech synthesizers," *arXiv preprint arXiv:2301.02111*, 2023.

[13] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Model. User-Adapted Interact.*, vol. 4, no. 4, pp. 253–278, 1994.