International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

TRESK-AI: An Adaptive, Resume-Conditioned Platform for AI-Driven Mock Interview Practice and Automated Performance Evaluation

Yash Galande
Dept. of Computer Science and Engineering
MIT School of Computing, MIT-ADT University
Pune, Maharashtra, India
yashgalande39@gmail.com

Akshat Jha
Dept. of Computer Science and Engineering
MIT School of Computing, MIT-ADT University
Pune, Maharashtra, India
akshatkumarjha108@gmail.com

Utkarsh Chavan
Dept. of Computer Science and Engineering
MIT School of Computing, MIT-ADT University
Pune, Maharashtra, India
chavanutkarsh50@gmail.com

Arjun Deshmukh
Dept. of Computer Science and Engineering
MIT School of Computing, MIT-ADT University
Pune, Maharashtra, India
arjundeshmukh857@gmail.com

Abhishekh Sarkar
Dept. of Computer Science and Engineering
MIT School of Computing, MIT-ADT University
Pune, Maharashtra, India
abhishek.sarkar@mituniversity.edu.in

Abstract*—TRESK-AI is an implemented full-stack platform for AI-driven mock interview practice; this paper reports its architecture, adaptive interviewing mechanism, and evaluation methodology as they exist in the deployed system rather than as originally proposed. Because the platform depends on third-party, largely free-tier large language model (LLM) endpoints with no uptime or rate-limit guarantees, its most consequential engineering contribution is an application-level reliability architecture – ordered multi-model failover through a gateway, a secondary direct-API fallback, response caching, and a fully deterministic rule-based fallback – that keeps question generation and answer evaluation available even when every upstream LLM call fails. Building on this foundation, the platform generates interview questions conditioned on a candidate's resume and target role, and adjusts follow-up question depth using a threshold on a documented, per-answer evaluation score that blends an LLM-judged holistic rating with a deterministic, transcript-derived heuristic. Feedback is expressed along multiple named dimensions rather than a single opaque number. The platform additionally supports voice-based interaction through browser-native speech recognition with a cloud speech-to-text fallback, an optional sandboxed coding-assessment module built on the Piston execution engine, and lightweight anti-cheating monitors based on window-focus tracking. During implementation review, we determined that the platform's on-screen gaze and stress indicators are client-side simulations rather than outputs of a genuine video-analysis pipeline; we report this finding directly and scope our claims accordingly (Section VI-E). We describe the system's technology stack, its resume-conditioned question-generation and evaluation mechanisms, and a concrete experimental protocol – covering adaptive-versus-static question generation, resume-conditioned-versus-generic questioning, and AI-versus-human-rater evaluation agreement – designed to be executed on this implementation. As a first, informal step toward that evaluation, we additionally report preliminary code-level findings executed as part of this revision: measured automated-test coverage and a dependency-vulnerability scan. No controlled human-subjects study has yet been conducted, so we do not report accuracy, correlation, or usability figures from that protocol; we instead specify the metrics, baselines, and threats to validity relevant to interpreting such a study once it is run.

***Index Terms*—mock interview systems, large language models, adaptive questioning, resume-conditioned generation, automated assessment, natural language processing, LLM reliability**

I. Introduction

Interview performance is difficult to practice in a way that produces useful, repeatable feedback. Peer mock interviews depend on the availability and interviewing skill of whoever is asked to help; static question banks cannot adapt to what a candidate has actually said; and one-on-one coaching does not scale to the number of candidates who would benefit from it. These constraints motivate software systems that can generate role-relevant questions and give structured feedback on a candidate's answers without requiring a human interviewer to be present for every session.

Large language models make parts of this feasible that were not feasible with earlier NLP pipelines: a general-purpose LLM, built on the transformer architecture [17] and exhibiting the instruction-following behavior documented for contemporary frontier models [18], can produce a fluent, role-appropriate interview question from a short prompt, sustain

enough context to ask a coherent follow-up, and produce a structured, multi-part critique of a candidate's answer in the same call that generated the question. What is less settled – and what motivates treating this as an engineering and research problem rather than a solved one – is how to combine LLM-based question generation and evaluation with the practical constraints of a real deployment: unreliable free-tier model availability, the cost of calling an LLM for every question and every evaluation, the risk of prompt injection from user-supplied resume and answer text, and the difficulty of validating that an LLM's evaluation of an answer is trustworthy.

This paper reports on TRESK-AI, a mock interview platform that we built and that is available as a working implementation, and revises our earlier description of the system – which consisted of internal project documentation and product-facing materials (a README and prior internal project reports) rather than a prior peer-reviewed publication of this work – to match what the implementation actually does. In preparing this revision we re-examined the codebase module by module rather than relying on the system's own documentation, and in doing so identified a specific and consequential gap between a previously claimed capability and the implementation, detailed in Section VI-E: the platform's on-screen "gaze tracking" and "stress detection" indicators are simulated rather than measured from video. We treat this as a finding to be reported rather than a detail to be smoothed over, because a paper that repeats a product description without checking the source code is not a reliable account of what was built.

The remainder of the paper is organized around what we were able to verify directly in the source, and in particular around two elements we regard as its most substantial and, in our assessment, most novel contributions: a three-tier LLM reliability cascade (multi-model failover through a gateway, a secondary direct API fallback, and a fully deterministic rule-based fallback) that is, to our knowledge, an underreported pattern at this scale in the applied interview-systems literature, and the implementation audit itself, including the negative gaze/stress finding. We also verify a documented evaluation score that blends an LLM judgment with a transcript-derived heuristic, a real (if simple) anti-cheating mechanism based on window-focus events, and a sandboxed code-execution path built on the Piston execution engine.

The major contributions of this work are:

1) A description of a working mock-interview platform's architecture and adaptive-interviewing mechanism, verified directly against its source code rather than against its documentation or marketing description.
2) A formalization of the platform's follow-up-question adaptivity as a threshold-gated mechanism on a per-answer evaluation score, together with an explicit statement of which parts of a more general adaptive-difficulty formulation are implemented and which would need to be added.
3) A documented multi-dimensional answer-scoring formula that combines an LLM-judged holistic score with a deterministic, transcript-derived heuristic score at fixed, disclosed weights, together with an analysis of what the heuristic component actually measures.
4) A resilience architecture for LLM-dependent question generation and evaluation, combining model-level failover, response caching, and a fully offline rule-based fallback path, positioned relative to recent work on LLM serving reliability.
5) An explicit implementation audit that separates genuinely implemented capabilities (resume-conditioned question generation, voice interaction, sandboxed code execution, focus-based anti-cheating, transcript-derived speech metrics) from a capability we found to be simulated rather than implemented (webcam-based gaze and stress analysis), together with a concrete, not-yet-executed experimental protocol for evaluating the implemented capabilities.

The remainder of this paper is organized as follows. Section II reviews related work and positions the research gap. Section III states the problem formulation and contributions precisely. Section IV describes the verified system architecture. Section V formalizes the adaptive interview methodology. Section VI describes the implementation in more detail, mapped to the repository structure. Section VII specifies the experimental setup, and Section VIII reports preliminary, code-level findings executed as part of this revision and discusses what a full evaluation is expected to show, since no human-subjects study has yet been executed. Section IX addresses security, privacy, and ethics; Section X states limitations and threats to validity, including the gaze/stress finding; Section XI outlines future work; Section XII concludes.

II. Related Work and Research Gap

A. AI-based and automated interview analysis systems

Applied systems for AI-assisted interviewing typically combine a question source with an automated scoring component, sometimes supplemented by webcam-based analysis of non-verbal behavior. A separate and more empirically mature line of work has studied automated judgment of interview performance directly from recorded video and audio rather than generation of questions. Tanveer et al. modeled the relationship between multimodal behavioral cues and human-rated interview performance, showing that features derived from speech and language content carry predictive signal for perceived interview quality [1]. Chen et al. extended automated interview judgment to a large, online-collected video corpus, illustrating both the feasibility and the noisiness of automatically scoring interview videos at scale [2]. Both systems are fundamentally analysis systems – they score a pre-existing recording – rather than generative, adaptive systems that decide what to ask next; this distinction matters because it means their reported reliability figures describe a different task from the one TRESK-AI's follow-up mechanism performs. More recent applied work has reported browser-based interview interfaces that pair video capture with automated scoring for employer-side screening [3], though published capability details for such systems are limited.

TABLE I
TRESK-AI POSITIONED AGAINST THE CLOSEST RELATED SYSTEMS IN §II-A–B.

System	Task	Adaptive	Resume-cond.	Validated
Tanveer et al. [1]	Analysis (post-hoc)	No	No	Yes
Chen et al. [2]	Analysis (post-hoc)	No	No	Yes
Karim et al. [4]	Selection (classify)	N/A	N/A	Yes
Hoque et al. [5]	Selection (fuzzy MCDM)	N/A	N/A	Yes
TRESK-AI (this paper)	Generation + scoring	**Yes** (§V-C)	**Yes**	**Not yet** (§VII)

B. Automated candidate and resume assessment

Resume-screening research has moved from keyword matching toward LLM-based profile evaluation. Karim et al. fine-tuned transformer models to classify software-engineering candidate profiles as suitable or unsuitable for a role [4], and follow-up work combined LLM-based scoring with fuzzy multi-criteria decision-making to address the subjectivity of human suitability judgments [5]. These systems perform selection (a binary or ranked suitability judgment) rather than question generation conditioned on a resume; TRESK-AI's resume pipeline uses the same class of LLM-based structured extraction but applies it to conditioning interview content rather than to a hiring decision, which is a materially different, and lower-stakes, use of the same underlying technique.

Table I summarizes this distinction alongside the empirical-validation status of each system, including TRESK-AI's own. TRESK-AI is, to our knowledge, the only system in this comparison set that is simultaneously generative, adaptive, and resume-conditioned – and, consistent with the honesty this paper aims for throughout, it is also the only one in the table without a completed empirical evaluation. We read Table I as motivating rather than undermining Section VII's protocol: the capability gap is real, but so is the validation gap, and closing the latter is a prerequisite for the former to be a defensible claim rather than an architectural description.

C. Reliability and fairness of LLM-based evaluation

Because TRESK-AI uses an LLM both to generate questions and to judge answers, two adjacent literatures are directly relevant. First, work on hallucination in large language models catalogues the causes, detection strategies, and mitigations for fluent but unsupported model output [13]; this motivates the platform's rule-based rubbish-response filter and JSON-schema validation layer, described in Section VI, which do not rely on the LLM to self-report when it has failed. Second, "LLM-as-a-judge" evaluation has been shown to correlate reasonably with human judgment on general instruction-following tasks [7], but to degrade specifically on tasks requiring specialized domain expertise, where agreement with subject-matter experts is markedly lower than on general tasks [8]. This second finding is a central input to Section VIII: it is a specific, literature-grounded reason to expect that TRESK-AI's technical-correctness scoring may be less reliable for narrow, expert-level technical questions than for general communication and completeness dimensions, and it is exactly the kind of claim that the AI-versus-human-rater experiment in Section VII is designed to test rather than assume.

Fairness is a related but distinct concern. Fabris et al. survey bias in algorithmic hiring broadly, covering sources of bias in training data, label definition, and feature selection, and conclude that whether algorithmic hiring is less biased than the human-only alternative remains an open, context-dependent question [6]. This motivates treating TRESK-AI's output as formative practice feedback rather than as a hiring signal, and motivates the fairness auditing gap identified in Section X, since no such audit has yet been performed on this platform.

D. Prompt engineering and LLM serving reliability

TRESK-AI's question generation and evaluation are implemented through prompted calls to general-purpose, non-fine-tuned LLMs, so prompt design materially affects behavior; Vatsal and Dubey survey prompting strategies and their differential effectiveness across NLP task types [9], consistent with the structured, role-and-context-conditioned prompt templates described in Section VI. Because the platform depends on third-party, largely free-tier LLM endpoints that are known to be unreliable in production, work on LLM serving resiliency is directly relevant: Qian et al. document that LLM serving infrastructure remains fragile in practice, with recovery from partial failures often taking minutes without dedicated fault-tolerance mechanisms [15]. TRESK-AI's model-gateway failover and caching design, described in Section IV, is best read as an application-level response to exactly this class of problem, implemented with ordinary web-application tools (an in-memory cache, a Redis-backed cache, and an ordered failover list) rather than with serving-infrastructure-level techniques.

E. Speech interaction and retrieval-augmented assistance

Automatic speech recognition is required for voice-based interview sessions; large-scale weakly supervised ASR models have demonstrated robust multilingual transcription across accents and noisy conditions [10], representing the general class of cloud ASR service to which TRESK-AI's server-side transcription fallback belongs, distinct from the browser-native recognition path that the platform uses as its primary mechanism (Section VI-E). Retrieval-augmented generation, in which a language model's output is grounded in retrieved external content rather than relying solely on parametric knowledge [16], is the general technique underlying the platform's separate conversational assistant module (Section VI); this module is architecturally adjacent to, but distinct from, the core interview pipeline evaluated in this paper.

F. Speech-derived and educational feedback systems

Automated essay and short-answer scoring research has developed approaches that generate explanatory feedback in addition to a score, addressing the criticism that automated scoring alone is not actionable for the learner [11]; TRESK-AI's per-answer feedback (strengths, improvements, and missing points, described in Section V-D) follows this general pattern. Educational chatbot research reports generally positive but heterogeneous learning outcomes that depend on instructional

design as much as on the underlying technology [12], which we take as a caution against assuming that conversational delivery alone improves interview-practice outcomes without direct measurement.

G. Secure execution of untrusted code

The optional coding-assessment module executes candidate-submitted code and therefore inherits the security requirements of any online-judge system. Kuo et al. analyze the attack surface of code-execution sandboxes, including denial-of-service and resource-exhaustion risks, and propose corresponding mitigation rules [14]; this directly informs the description of Piston-based execution and its resource limits in Section VI.

H. Research gap

Read together, this literature indicates three specific gaps relevant to TRESK-AI. First, systems that generate adaptive, resume-conditioned interview content are less studied empirically than systems that score pre-recorded interviews, so the reliability figures available in the literature (Sections II-A, II-C) do not directly transfer to a generation-and-follow-up setting and must be measured separately, which Section VII is designed to do. Second, published descriptions of AI interview platforms rarely report an implementation audit that distinguishes marketed features from verified code behavior; this paper's central methodological contribution is doing so explicitly for one platform, including a negative finding (Section III, Section X). Third, work on LLM serving reliability has focused on infrastructure-level techniques for large-scale deployments [15]; the application-level failover and caching pattern used in TRESK-AI is a smaller-scale but directly-relevant instance of the same underlying problem, and its effectiveness (in terms of question-generation success rate and latency) is measurable on this system but has not yet been fully measured under live conditions, which we identify as remaining future work in Section XI.

III. Problem Formulation and Contributions

The problem this paper addresses has two parts. The first is a systems problem: how should resume-conditioned, adaptive interview question generation and multi-dimensional answer evaluation be implemented given that the underlying LLM calls are made against third-party endpoints with no uptime or rate-limit guarantees, particularly when those endpoints are free-tier services chosen for cost reasons rather than paid, higher-reliability alternatives? The second is a reporting problem specific to this paper: given that an earlier description of this platform was written without a full source-code audit, what does the platform actually do, and which previously implied capabilities does it not have?

Formally, let R denote a candidate's parsed resume, J the selected job role and interview type, H_t the interview history up to turn t (prior question-answer pairs), A_t the candidate's answer at turn t, and S_t the evaluation score assigned to A_t. The platform's question-generation behavior can be described as a function

$$Q_1 = f_{init}(R, J), \qquad Q_{t+1} = f_{adapt}(H_t, A_t, S_t), \qquad (1)$$

where f_{init} produces an initial batch of role- and resume-conditioned questions and f_{adapt} produces a follow-up. Equation (1) is a general formulation; Section V-C states precisely which part of it corresponds to implemented behavior (a two-way branch in f_{adapt} conditioned on a threshold on S_t) and which part would require additional development (a persistent, multi-level difficulty state D_t that is adjusted incrementally across an entire session, which is not currently maintained).

Equation (1) also gives a precise way to state a property of the reliability cascade described in Section IV and Section VI: because f_{init} and f_{adapt} are each ultimately backed by the same three-tier fallback chain, the wall-clock time to exhaust the OpenRouter gateway stage alone – before a direct Gemini attempt or the rule-based fallback is reached – is bounded by

$$T_{gw} = N_{gw}\,\tau_{gw}, \qquad (2)$$

where N_{gw} is the number of models in the gateway's ordered failover list and τ_{gw} is the per-model timeout enforced in `openrouter.js`. At the snapshot reviewed for this revision, $N_{gw} = 7$ (an Nvidia Nemotron variant as primary, two further Nemotron variants, two Google Gemma variants, one OpenAI `gpt-oss` variant, and one Poolside model, tried in that order) and $\tau_{gw} = 30$ s, giving $T_{gw} = 210$ s under the current configuration. This bound applies only to the gateway stage: on re-inspection of the fallback path for this revision, we found that the secondary direct Gemini call issues a plain `fetch` request with no explicit `AbortController` or timeout, so it does not inherit a comparable enforced bound, and the platform's only source of a hard, source-verified latency guarantee is therefore the fully deterministic rule-based fallback reached once both LLM stages have failed or the Gemini stage is skipped (e.g., no API key configured). We surface this asymmetry here, as a direct consequence of formalizing Eq. (1), because it is exactly the kind of source-level detail that a purely descriptive account of the three-tier cascade would not have forced us to check.

Given this formulation, the contributions listed in Section I follow directly: Contribution 1 establishes what f_{init} and f_{adapt} actually compute; Contribution 2 states precisely how much of a general adaptive-difficulty model is and is not implemented; Contribution 3 specifies how S_t is computed; Contribution 4 describes the reliability mechanism that makes calls to f_{init} and f_{adapt} tolerant of upstream LLM failures, now stated quantitatively via Eq. (2); and Contribution 5 is the audit itself, including the negative finding about the webcam-based indicators, which are not part of S_t's technical-correctness or completeness computation but are blended, in one place, into the persisted stress-score field (Section V-D).

IV. System Architecture

Fig. 1 shows the verified system architecture, organized into the seven layers used consistently throughout this paper. Every

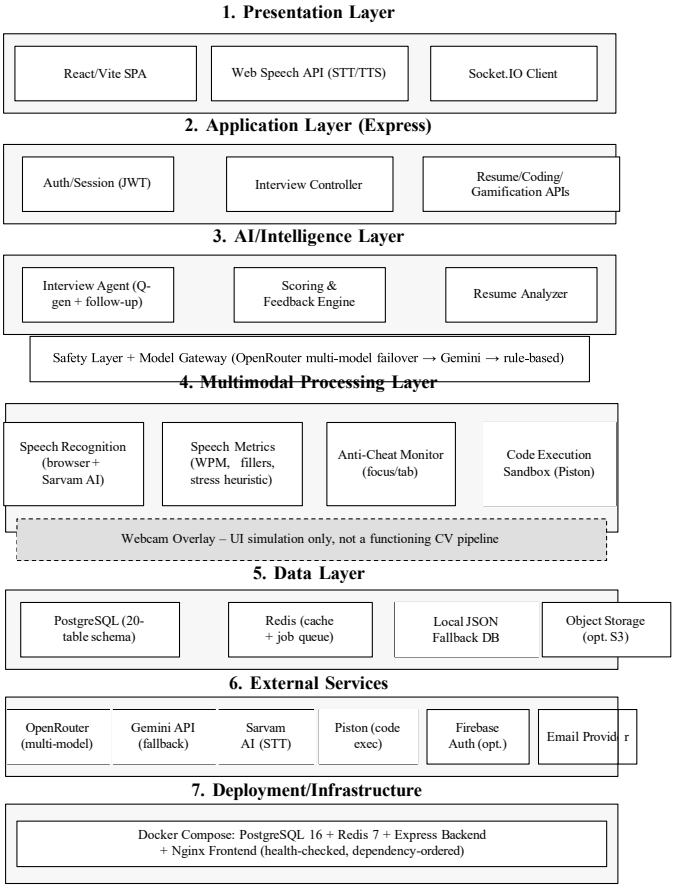


Fig. 1. Verified system architecture of TRESK-AI. Solid-bordered components were located directly in the repository source; the dashed, gray-shaded component is a client-side visual simulation with no underlying analysis pipeline (Section III, Section VI-E).

component shown was located in the repository's source code; the one component drawn with a dashed border and shaded differently – the webcam overlay in the Multimodal Processing Layer – is explicitly marked because it is a client-side visual simulation rather than a functioning analysis pipeline, as established in Section III and detailed in Section VI-E.

Presentation layer. The client is a React single-page application built with Vite. Voice interaction uses the browser's native `SpeechRecognition` (`webkitSpeechRecognition`) API for speech-to-text and the `SpeechSynthesis` API for reading questions aloud, avoiding a network round trip for the primary transcription path. A Socket.IO client maintains a real-time connection for session updates.

Application layer. The backend is a single Express.js application organized into route-and-controller modules (auth, interview, resume, coding, gamification, analytics, admin, notification) rather than as independently deployed microservices; this is stated explicitly because an earlier description of the system implied a more service-oriented decomposition than the code exhibits. Authentication issues a short-lived JWT access token together with an httpOnly, rotating refresh-token cookie, with refresh sessions tracked in a dedicated database table; passwords are hashed with bcrypt.

AI/Intelligence layer. Four LLM-backed services – the interview agent, the scoring/feedback engine, the resume analyzer, and a recommendation engine – all route their model calls through a shared gateway module rather than calling a model provider directly. This gateway (Section IV, Section VI) implements ordered multi-model failover across several free-tier models on OpenRouter, an in-memory plus Redis-backed response cache, and a prompt-injection and output-validation safety layer applied to all user-supplied text before it reaches a prompt.

Multimodal processing layer. This layer is the one requiring the most care in description. Speech recognition, transcript-derived words-per-minute, and filler-word counting are genuinely computed from the candidate's actual transcript and timing data. A window-focus-based anti-cheating monitor and a Piston-backed code-execution sandbox are both real, server-verified mechanisms. The webcam overlay – drawn with a dashed border in Fig. 1 – renders an animated tracking graphic over the candidate's live camera feed, but the animation is generated from a sinusoidal function of wall-clock time, not from any per-frame image analysis, and the eye-contact and stress values shown to the candidate during the session are produced by a bounded random walk and a counter that increments in lockstep with itself, respectively (Section VI-E gives the exact mechanism).

Data layer. PostgreSQL 16 is the primary store, with a schema of approximately twenty tables covering users, interview sessions, resumes, coding submissions, questions, daily challenges, refresh sessions, audit logs, AI cost logs, feature flags, learning paths, and notifications. Redis backs both the L2 response cache and a BullMQ job queue (Section VI). When PostgreSQL, Redis, or the configured AI providers are unreachable, the backend can fall back to a local JSON file database and a fully deterministic, template-based question synthesizer, allowing the application to remain functional in a degraded "demo" mode; this fallback path is a real, distinguishing engineering feature of the deployment, not merely a development convenience, since it is reachable in production via an explicit configuration flag.

External services. OpenRouter is the primary LLM access point; the Gemini API is called directly as a secondary fallback when OpenRouter's entire model list is exhausted; Sarvam AI provides server-side, multilingual speech-to-text as an alternative to the browser's native recognizer; Piston executes untrusted code in isolated containers; Firebase supports optional social login; and an email-provider abstraction supports transactional email with a console-logging fallback for local development.

Deployment/infrastructure. A four-service Docker Compose configuration (PostgreSQL, Redis, the Express backend, and an Nginx-served frontend build) with health checks and explicit startup dependency ordering is the verified deployment path. No orchestration framework beyond Docker Compose (e.g., Kubernetes) was found in the repository, and no evidence of a microservices decomposition was found; both claims

are stated explicitly to avoid the overclaiming this paper is specifically trying to correct.

V. Adaptive Interview Methodology

A. Resume processing

Uploaded resumes (PDF) are parsed to raw text using `pdf-parse` and then passed, truncated to a bounded length, to an LLM prompt that requests a structured JSON object (name, contact details, skills, experience, projects, education, and an inferred target role) at a low sampling temperature ($T = 0.1$) chosen to favor consistency over creative variation. This differs from a rule-based, taxonomy-driven extraction pipeline: the platform does not maintain a curated skills taxonomy or use a named-entity-recognition model for this step, and the accuracy of the extracted fields is therefore bounded by the underlying LLM's extraction reliability rather than by a separately validated NLP component. We state this explicitly because a taxonomy-based description would overstate the maturity of this step relative to what is implemented.

B. Question generation

Given a parsed resume, role, company, interview type, and difficulty label, the interview agent constructs a single prompt requesting exactly five questions as a JSON array, at a higher sampling temperature ($T = 0.75$) intended to favor variety across sessions. When resume text of sufficient length is present, the prompt instructs the model to reference specific projects, technologies, or claimed metrics from the resume rather than asking role-generic questions; when no usable resume text is present, the same prompt template omits this instruction and falls back to role- and type-conditioned generation only. This corresponds to $f_{init}(R, J)$ in Eq. (1). If the primary model call fails validation or returns fewer than three questions, the controller falls back first to a direct Gemini API call and, if that also fails, to a fully deterministic, template-based question synthesizer that interpolates resume fields (or generic placeholders, in the absence of a resume) into fixed sentence templates – described in the interview controller as "rule-based fallback questions." This three-tier structure means that the platform can produce some interview question in the complete absence of any working LLM connection, at the cost of that question no longer being adaptively generated in the sense implied by Eq. (1).

C. Adaptive follow-up (difficulty) mechanism

The implemented adaptivity operates at the level of a single follow-up decision rather than as a persistent difficulty state carried across an entire session. After an answer A_t is scored, producing $S_t \in [0, 100]$ (Section V-D), the follow-up generator selects one of two prompt instructions:

$$\text{depth}(S_t) = \begin{cases} \text{deeper / more technical}, & S_t \geq \tau \\ \text{clarifying / supportive}, & S_t < \tau, \end{cases} \quad (3)$$

with $\tau = 80$ in the current implementation. This threshold is a manually chosen heuristic constant, not a value derived from experimental calibration against human-interviewer behavior; we state this explicitly per the concern raised in Section VII regarding unvalidated thresholds. Equation (3) is a two-branch instance of f_{adapt} from Eq. (1): it conditions the next question's framing on S_t, but it does not maintain an explicit, incrementing difficulty variable D_t across turns. A more general formulation such as

$$D_{t+1} = \begin{cases} D_t + 1, & S_t \geq \tau_{high} \\ D_t - 1, & S_t \leq \tau_{low} \\ D_t, & \text{otherwise} \end{cases} \quad (4)$$

is a proposed extension consistent with the platform's existing evaluation signal, not a description of current behavior; implementing it would require persisting D_t per session and mapping its value onto the discrete difficulty labels (Beginner/Intermediate/Advanced/Expert) already used by the initial question-generation prompt. We list this as near-term future work in Section XI rather than describing it as implemented. Algorithm 1 summarizes the per-turn control flow as implemented, including the three-tier fallback path from Section V-B.

Algorithm 1 Per-turn interview control flow (as implemented)

Require: resume R, role/type J, history H_t

```
if t = 0 then
    Q_1 ← GENERATEQUESTIONS(R, J)        ▷ 5 questions, batch
    on failure: try Gemini fallback, then rule-based fallback
else
    A_t ← candidate's submitted answer to Q_t
    S_t ← EVALUATEANSWER(Q_t, A_t, J)    ▷ Section V-D
    if S_t ≥ τ then
        Q_{t+1} ← GENERATEFOLLOWUP(Q_t, A_t, S_t, "deeper")
    else
        Q_{t+1} ← GENERATEFOLLOWUP(Q_t, A_t, S_t, "supportive")
    end if
end if
Persist (Q_t, A_t, S_t) to interview_sessions transcript
until session question budget exhausted
Compile session feedback (Section V-E)
```

D. Answer evaluation

Each submitted answer first passes through a deterministic rubbish-response filter – a small set of regular-expression and word-count rules that detect empty, one-word, greeting-only, or keyboard-mash responses – before any LLM call is made; responses flagged this way are assigned a fixed low score without incurring an LLM call, which is both a cost-control measure and a defense against wasting an evaluation call

on unanswerable input. For substantive answers, the platform computes a final score as

$$S_t = \text{round}\ \left(0.7\, S_t^{LLM} + 0.3\, S_t^{heur}\right), \quad (5)$$

where S_t^{LLM} is an `overallScore` $\in [0, 100]$ returned directly by the LLM as part of a structured evaluation object that also includes separately reported `technicalScore`, `communicationScore`, and `completenessScore` fields, and

$$S_t^{heur} = 0.6\, T_t^{heur} + 0.4\, F_t^{heur}, \quad (6)$$

where T_t^{heur} is a word-count-bucketed proxy for technical depth (fixed values of 0, 45, 60, 85, or 95 depending on answer length thresholds, with 0 assigned to rubbish-filtered responses) and F_t^{heur} is 100 minus a penalty proportional to filler-word density, clamped to $[40, 100]$. We report Eq. (6) in full because T_t^{heur} is, by construction, a length heuristic rather than a technical-correctness measure, and this is a meaningful limitation of the 30% non-LLM component of Eq. (5) that should be stated plainly rather than left implicit in a higher-level description such as "technical correctness scoring." The weights 0.7/0.3 and 0.6/0.4 are fixed constants in the implementation, not experimentally fitted parameters; Section VII-D proposes a protocol for evaluating whether S_t agrees with independent human ratings, which would be a prerequisite for recalibrating them.

A related, separately computed quantity is a speech-derived stress proxy,

$$\text{stress}_t^{speech} = \min(100,\ \max(10,\ 12\ \text{fillers}_t + \delta_t)), \quad (7)$$

where fillers_t is the filler-word count for the answer and $\delta_t = 30$ if the transcript's words-per-minute rate is outside a $[85, 150]$ band and 10 otherwise. This is a heuristic derived entirely from speech-timing and lexical signals; when a client-supplied webcam-derived stress value is also present in the request, the persisted stress score is the mean of the two. Because the webcam-derived value is, per Section VI-E, not a genuine measurement, the persisted stress score can be partially influenced by a non-informative input; we flag this explicitly as a limitation in Section X and recommend that the webcam-derived term be removed from Eq. (7) until a genuine visual pipeline replaces it.

E. Feedback generation

At the end of a session, the platform issues a further LLM call that receives the full scorecard (including the eye-contact and stress fields discussed above) and, where available, the complete question-answer transcript, and requests a structured object containing an overall verdict, a numeric hiring-likelihood estimate, a short natural-language coaching paragraph, and three named strengths and three named improvement areas. A separate recommendation-engine call produces a study plan and suggested certifications. Because the scorecard passed into this prompt includes the eye-contact and (partially simulated) stress fields, the generated natural-language feedback can in principle reference these fields (for example, praising "strong eye contact") even though no such measurement was actually made (Section VI-E); this is a second, downstream consequence of the finding in Section III and is listed as a required near-term fix in Section XI.

VI. System Implementation

A. Frontend

The frontend (`frontend/src`) is a React 18 application built with Vite, styled with Tailwind CSS, using Framer Motion for animation, Recharts for dashboard visualizations, and a Socket.IO client for real-time updates. Firebase is included as a dependency and is used for optional social authentication; primary authentication is handled by the backend's own JWT/cookie flow.

B. Backend

The backend (`backend/src`) is an Express 4 application organized into per-feature modules (auth, interview, resume, coding, gamification, analytics, admin, notification, ai) under `modules/`, each exporting an Express router and a controller. Structured logging uses Pino; request validation uses Joi; security middleware includes Helmet and `express-rate-limit`; database access uses the `pg` driver directly against parameterized SQL queries defined in each controller, with no object-relational mapper.

A Jest test suite (`backend/src/tests`, split into unit and integration) targets the AI services, utility functions, middleware, and the authentication controller specifically, per the project's `package.json` coverage configuration. Running this suite (`npm run test:coverage`) against the snapshot reviewed for this revision passed all 12 test suites (142 individual tests, 0 failures) and reported aggregate coverage, restricted to the files the suite targets, of 36.6% statements, 34.0% branches, 31.5% functions, and 37.1% lines. This aggregate figure is uneven across the targeted modules and is worth reporting at that granularity rather than as a single number: `sanitizePromptInput.js` and `validate.js` are fully covered (100% statements); `safetyLayer.js` (98.2%) and `scoringEngine.js` (91.1%) are well exercised; `openrouter.js`, which implements the failover cascade central to Contribution 4, is only 6.3% covered, meaning the retry, caching, and failover logic itself is almost entirely untested; and `interviewAgent.js` and `resumeAnalyzer.js` – two of the five core AI-service modules, including the one responsible for f_{init} and f_{adapt} in Eq. (1) – report 0% coverage despite being listed in the suite's own coverage target configuration. We report these module-level figures, rather than only the aggregate, because the aggregate alone would understate how little of the platform's most novel mechanism (the failover cascade) and its core adaptive-generation logic are currently exercised by automated tests; closing this gap, starting with `openrouter.js` and `interviewAgent.js`, is listed as near-term work in Section XI.

C. AI services

Five modules under `services/ai` implement the platform's LLM-backed behavior: `interviewAgent.js` (question generation and follow-up), `scoringEngine.js` (answer evaluation and, separately, code review), `feedbackEngine.js` (session-level feedback), `resumeAnalyzer.js` (resume parsing, ATS scoring, and job-description analysis), and `safetyLayer.js` (input/output validation). All model calls are routed through `openrouter.js`, which implements: (i) an ordered failover list currently including several free-tier models (an Nvidia Nemotron variant as primary, with Google Gemma, OpenAI `gpt-oss`, additional Nemotron variants, and a Poolside model as ordered alternatives), (ii) a 200-entry, 30-minute-TTL in-memory LRU cache, (iii) a Redis-backed secondary cache, and (iv) response parsing that strips reasoning-model `<think>` blocks and markdown code fences before attempting JSON parsing, with three progressively more permissive fallback parsing strategies if strict parsing fails. `safetyLayer.js` independently applies pattern-based detection for prompt-injection phrasing, SQL-injection fragments, and script tags to user-supplied text before it is interpolated into a prompt, and validates that LLM output is not a refusal and, where a JSON schema is expected, contains the required keys. This safety layer and the resilience mechanism in `openrouter.js` are two of the more substantial pieces of original engineering in the codebase and are, in our assessment, better evidence of the project's technical depth than any individual feature-level claim.

D. Database and asynchronous processing

The primary schema (`schema.sql`, under `backend/src/config/`) defines approximately twenty PostgreSQL tables, including `interview_sessions`, `replay_events`, `resumes`, `coding_submissions`, `questions`, `daily_challenges`, `refresh_sessions`, `audit_logs`, `ai_cost_logs`, `ai_prompts`, `feature_flags`, `learning_paths`, `user_skill_profiles`, and `notifications`. Post-interview feedback compilation and multi-stage resume ATS parsing are dispatched as BullMQ jobs to a Redis-backed queue (`queues/aiQueue.js`) and processed by a separate worker process (`workers/aiWorker.js`), with three retry attempts and exponential backoff on job failure; this decouples expensive, multi-call LLM workloads from the synchronous request/response cycle. When PostgreSQL or Redis is unreachable, the affected controllers fall back to an in-memory session map and, for persisted data, a local JSON file (`local_db.json`), gated by an explicit `ALLOW_DEMO_AUTH` configuration flag.

E. Voice and multimodal components

Voice input during a live session is transcribed primarily by the browser's native `SpeechRecognition` API, with a server-side endpoint (`POST /api/ai/transcribe`) available as an alternative path that forwards audio to Sarvam AI's `saaras:v3` model, which supports multiple Indian languages in addition to English. Words-per-minute and filler-word count are computed directly from the transcript and an elapsed-time value; both are genuine, deterministic computations over real input.

The webcam-based indicators require a precise, complete description, since this is the single most consequential finding of this revision. The frontend requests camera access and displays the live video feed, and draws an animated bounding-box-and-pupil overlay on a separate canvas layer, but the overlay's motion is generated by sine and cosine functions of the current timestamp and is not derived from any per-frame pixel or facial-landmark analysis; the displayed "eye contact" percentage is computed from two counters that are incremented together on every animation frame during which the candidate is speaking, so that the ratio is definitionally at or near 100% regardless of the candidate's actual gaze behavior; the displayed "stress" percentage performs a small bounded random walk independent of any sensor input; and the displayed dominant "emotion" is a hard-coded string. None of these three values should be read as a measurement, and we did not find any facial-landmark, gaze-estimation, or emotion-classification model anywhere in the repository. Every other reference to this finding elsewhere in the paper (Sections I, IV, V-D–E, IX, X) is a deliberately short cross-reference back to this paragraph, so that the finding is stated once, completely, in the place where the underlying mechanism is described, rather than re-argued at each occurrence.

F. Coding assessment

The optional coding module executes candidate-submitted code against the public Piston API (`emkc.org/api/v2/piston`) with a run timeout of 5000 ms, a compile timeout of 10000 ms, and a run-time memory limit of 256 MB. The codebase explicitly documents the removal of a previously used in-process JavaScript sandbox (`vm2`) due to known remote-code-execution vulnerabilities (CVE-2023-29017, CVE-2023-37466), replaced by Piston's container-based isolation; this is stated in the source as a security rationale, not merely a feature change. If Piston is unreachable, the controller falls back to asking the LLM itself to simulate program execution and report per-test-case results as structured JSON. We flag this fallback explicitly as a reliability weakness rather than a redundant safeguard: an LLM's simulated prediction of a program's output is not equivalent to actually executing it, and a pass/fail verdict produced this way should not be treated as a verified correctness result.

VII. Experimental Setup

No controlled human-subjects evaluation has yet been conducted on this platform. This section specifies the protocol we propose to run, distinguishing it clearly from the preliminary findings and discussion of expected outcomes in Section VIII.

A. Participants

We propose a volunteer sample of computer-science students or early-career engineers (target $n \geq 20$ for the within-subjects comparisons in Experiments 1–2, and a separate set of at least three qualified human raters for Experiment 3) recruited from the authors' institution. No participants have been recruited at the time of writing; no results from this section should be read as having been collected.

B. Baselines

Two baselines are defined against the proposed condition (resume-conditioned adaptive interviewing, as implemented and described in Section V): Baseline 1 – static question bank, a fixed sequence of role-typed questions drawn from the platform's own seeded question table with adaptivity disabled; and Baseline 2 – generic LLM interview, the same question-generation prompt with the resume-conditioning block removed, isolating the effect of resume conditioning from the effect of using an LLM at all.

C. Metrics

Experiment 1 (adaptive vs. static). Human-rated relevance and difficulty-appropriateness (5-point Likert), candidate-reported satisfaction (Likert), and session completion rate. **Experiment 2 (resume-conditioned vs. generic).** Independent human evaluators, blind to condition, rate relevance, personalization, and technical appropriateness (5-point Likert) for matched question sets. **Experiment 3 (AI vs. human evaluation agreement).** The same set of recorded candidate answers is scored independently by the platform (Eq. (5)) and by human raters using the same four named dimensions; agreement is summarized with mean absolute error, Pearson and Spearman correlation, and Cohen's weighted kappa, computed separately for the LLM-only component S_t^{LLM} and the blended score S_t to isolate the effect of the heuristic term in Eq. (6). **Experiment 4 (system performance).** End-to-end latency from answer submission to returned score, decomposed into rubbish-filter time, LLM call time (including failover attempts, if any), and database write time; reported as mean, median, and 95th-percentile values over a fixed number of repeated calls against the live endpoints. **Experiment 5 (usability).** System Usability Scale (SUS) administered after a complete mock-interview session, reported with sample size and participant profile alongside the score.

D. Protocol

For Experiments 1–2, each participant completes matched sessions under both the proposed condition and the relevant baseline, in counterbalanced order, to control for practice effects; question sets shown to human raters in Experiment 2 are anonymized with respect to condition. For Experiment 3, human raters are calibrated on a small shared subset before independent rating of the remaining answers. For Experiment 4, latency is measured under normal operating conditions against the live OpenRouter endpoint rather than against a mocked LLM response, since failover and cache behavior are part of what is being measured.

VIII. Preliminary Findings and Discussion of Expected Outcomes

A. Preliminary, code-level findings (executed)

Sections VII-A–E describe a human-subjects protocol that has not yet been run. Independently of that protocol, and as part of preparing this revision, we executed two code-level checks that do not require participants and that bear directly on claims made elsewhere in this paper. We report them here, explicitly labeled as preliminary and informal rather than as a substitute for Section VII's confirmatory study, consistent with the reporting standard the rest of this paper tries to hold itself to.

Automated test coverage. As reported in Section VI-B, the existing Jest suite passes in full (142/142 tests) but leaves `openrouter.js` – the module implementing the failover cascade behind Contribution 4 – almost entirely untested (6.3% statement coverage), and leaves `interviewAgent.js` and `resumeAnalyzer.js` at 0% despite being named in the suite's own coverage configuration. This is a finding about test scope, not about runtime correctness: passing tests on a narrow, well-covered subset of modules (`safetyLayer.js`, `scoringEngine.js`, the utility functions) says little about the reliability of the modules this paper's Contribution 4 is centered on.

Dependency vulnerability scan. Running `npm audit` against the backend dependency tree reported zero known vulnerabilities. The same scan against the frontend dependency tree reported two moderate-severity advisories, both in `react-router` (an open-redirect issue via a backslash in `<Link>`/`useNavigate`, and an arbitrary-constructor-injection issue via `deserializeErrors()` during server-side-rendering hydration), both with an automated fix available via `npm audit fix` at the time of this scan. This directly closes part of the gap identified in Section IX-B: the platform did not previously have any documented dependency-scan result, and now has one, even though the result itself (two fixable, moderate-severity, frontend-only advisories) is a modest finding rather than a clean bill of health.

We did not attempt Experiment 4's live-endpoint latency measurement as part of this revision, since it requires authenticated access to the OpenRouter and Gemini endpoints under realistic load and is properly a human-subjects-adjacent systems experiment rather than a one-off code check; Eq. (2)'s analytically derived bound is offered in Section III as a stated, falsifiable expectation for that future measurement, not as a substitute for it.

B. Expected outcomes of the not-yet-executed protocol

Because Sections VII-A–E have not yet been executed, this subsection does not report results for Experiments 1, 2, 3, or 5; it states what the mechanism described in Sections V–VI predicts, so that a future execution of the protocol has a stated hypothesis to confirm or refute rather than being interpreted post hoc.

Given the literature-grounded finding that LLM-as-judge agreement with human raters degrades on narrow technical

tasks relative to general tasks [8], we would expect Experiment 3 to show higher agreement (lower MAE, higher correlation) on the communication and completeness dimensions than on the technical-correctness dimension, and we would expect the blended score S_t (Eq. (5)) to track human ratings of answer length and fluency more closely than human ratings of technical depth, precisely because T_t^{heur} in Eq. (6) is a length proxy rather than a correctness measure. A result contrary to this expectation – for example, uniformly high agreement across all four dimensions – would itself be a useful and reportable finding, since it would suggest that the length heuristic is a better proxy for perceived technical quality than its construction would suggest, at least for the population sampled.

For Experiment 1, the specific mechanism in Eq. (3) – a single threshold on a blended score, applied only to follow-up framing rather than to a persistent difficulty state – predicts a modest rather than large improvement over the static baseline in difficulty-appropriateness ratings, since the platform does not currently adjust the difficulty of the initial five questions within a session based on in-session performance (Section V-C). We would not expect this experiment to show the same magnitude of benefit that a fully stateful difficulty model (Eq. (4)) might produce, and the experiment as designed cannot distinguish "adaptive follow-up framing helps" from "adaptive difficulty scaling helps," since only the former is implemented. This is stated here so that a future positive result is not over-interpreted as evidence for a mechanism the system does not currently have.

For Experiment 4, the three-tier failover chain in `openrouter.js` predicts a bimodal latency distribution, sharpened by the bound derived in Eq. (2): fast responses when the primary or an early failover model succeeds, and a smaller number of substantially slower responses – bounded at roughly $T_{gw} = 210$ s before the unbounded Gemini stage is even attempted – when several models are tried in sequence before success. Reporting only a mean latency would obscure this; we specify percentile reporting in Section VII-C for this reason, and the asymmetry noted in Section III between the gateway's enforced timeout and the Gemini fallback's unenforced one is itself now a specific, testable prediction for that future measurement: we would expect the tail of the latency distribution to be dominated by sessions that reach the Gemini stage.

IX. Security, Privacy and Ethical Considerations

A. Implemented mechanisms

Authentication uses bcrypt-hashed passwords, short-lived JWT access tokens, and httpOnly, rotating refresh-token cookies tracked in a dedicated database table, allowing individual sessions to be revoked. Helmet sets standard security headers; `express-rate-limit` applies per-route rate limiting; Joi schemas validate request bodies; all database access uses parameterized queries via the `pg` driver, which is a structural defense against SQL injection independent of the additional pattern-based SQL-injection check in `safetyLayer.js`. User-supplied text destined for an LLM prompt is passed through `sanitizePromptInput` (structural-character stripping and length capping) and, on relevant routes, the `promptGuardMiddleware` pattern-based injection filter. An `audit_logs` table exists in the schema for recording security-relevant events.

B. Recommended production safeguards not currently implemented

As reported in Section VIII-A, a dependency-vulnerability scan is no longer an unaddressed gap: `npm audit` against the snapshot reviewed for this revision reports zero vulnerabilities in the backend dependency tree and two moderate-severity, auto-fixable advisories (both in `react-router`) in the frontend dependency tree. We did not, however, find evidence that this scan is run automatically as part of a CI pipeline, so it should be read as a one-time, manual finding rather than a continuous safeguard; automating it (e.g., via a scheduled `npm audit` or a hosted dependency-scanning service in CI) remains a recommended addition. We also did not find a formal data-retention policy, an automated deletion job for resumes and transcripts, or a documented incident-response process on re-inspection for this revision; these remain absent and are standard recommendations for a system handling resumes, interview transcripts, and (for paid tiers, if enabled) payment-adjacent data.

C. Privacy considerations specific to this platform

Resumes and interview transcripts are personal data under most applicable privacy frameworks. Because the webcam feed is captured and displayed to the candidate (Section VI-E), even though it is not currently analyzed, any future implementation of a genuine gaze- or emotion-analysis pipeline would need explicit, informed consent separate from general platform terms of use, a defined retention period for any derived biometric signal, and a mechanism for the candidate to opt out of video capture entirely while still completing a text- or audio-only interview session; none of this consent infrastructure currently exists because the corresponding analysis capability does not exist.

X. Limitations and Threats to Validity

Gaze/stress simulation. As established in Section VI-E, the platform's webcam-based eye-contact and stress indicators are not derived from image analysis; any prior description of TRESK-AI as a "multimodal" or "biometric" interview-analysis system overstated this specific capability. We correct that here directly, as a limitation with a specific downstream effect on Eq. (7) and the feedback text in Section V-E, rather than treating it as a footnote.

Heuristic technical-correctness proxy. The 0.3-weighted heuristic component of Eq. (5) includes a technical-depth term that is, by construction, a function of answer length. This is disclosed in Section V-D but bears repeating as a limitation: a long, fluent, technically shallow answer can receive a higher blended score than a short, precise, technically strong one.

Unvalidated threshold. $\tau = 80$ in Eq. (3) is a manually chosen constant with no reported calibration against human-interviewer follow-up behavior.

No completed human-subjects evaluation. Section VII is a protocol, not a completed study; all claims about relevance, personalization benefit, or evaluation agreement are hypotheses pending the experiments in Section VII, not findings. The preliminary findings in Section VIII-A are code-level and do not substitute for it.

Free-tier LLM dependency. The primary model list in `openrouter.js` consists entirely of free-tier models at the time of writing, which are more likely to be rate-limited, deprecated, or degraded without notice than paid frontier models; this is a direct threat to the external validity of any latency or quality measurement taken at a single point in time, since free-tier model availability is known to fluctuate [15].

LLM-simulated code execution fallback. As described in Section VI, when Piston is unreachable the coding module's pass/fail verdict is produced by asking the LLM to predict execution results rather than by executing the code; any evaluation of the coding module's correctness must account for whether this fallback path was active.

Fairness auditing gap. No audit of question or score variation across resumes differing in protected-characteristic-correlated features has been performed on this platform, consistent with the general risk described in the algorithmic-hiring fairness literature [6].

Speech recognition and language coverage. The primary browser-based transcription path depends on browser support for the Web Speech API (notably absent or limited in some browsers), and the fallback Sarvam AI path, while multilingual, has not been evaluated for transcription accuracy on this platform's specific audio pipeline.

Generalizability. All implementation details in this paper reflect a single snapshot of the repository at the time of review; the model list, thresholds, and fallback behavior described are configuration-dependent and may change without a corresponding change to this paper. The test-coverage and dependency-scan figures in Section VIII-A are a further instance of this: both are properties of a specific commit and dependency lockfile state and will drift as the codebase changes.

XI. Future Work

Correcting the stress-score contamination. The clearest, most immediate fix identified by this audit is removing the client-supplied webcam-derived term from Eq. (7) and from the scorecard passed into the feedback-generation prompt (Section V-E) until a genuine visual-analysis pipeline exists, so that the persisted stress score and the generated feedback text do not implicitly claim a measurement that was not made.

A genuine, consent-gated visual pipeline. If webcam-based analysis is to be implemented, we recommend a validated facial-landmark and gaze-estimation approach (e.g., an off-the-shelf model such as MediaPipe Face Mesh) with an explicit accuracy evaluation against human-labeled ground truth, opt-in consent separate from general terms of use, a bounded retention period for any derived signal, and a documented false-positive rate before any "stress" or "confidence" label is shown to a candidate.

Stateful adaptive difficulty. Implementing Eq. (4) as a persisted, session-level difficulty variable that also affects the initial question batch (not only follow-ups), with τ_{high} and τ_{low} calibrated against Experiment 1 data rather than chosen manually.

Executing the proposed evaluation. Sections VII–VIII exist specifically to be executed; doing so, and using the results to recalibrate the weights in Eqs. (5)–(6), is the highest-priority next step for this line of work.

Removing or clearly labeling the LLM-simulated code execution fallback, given the correctness risk described in Section X, or replacing it with a self-hosted execution fallback to reduce dependence on the public Piston endpoint's availability.

Fairness audit. A structured audit of question and score variation across resumes varying in name, institution, and other proxies for protected characteristics, following established algorithmic-hiring fairness methodology [6].

XII. Conclusion

This paper re-examined TRESK-AI, a working mock-interview platform, directly against its source code rather than against its own documentation, and revised the platform's description accordingly. The verified implementation includes resume-conditioned question generation, a threshold-gated follow-up adaptivity mechanism, a disclosed blended scoring formula combining an LLM judgment with a transcript-derived heuristic, a three-tier LLM reliability cascade now bounded quantitatively for its gateway stage (Eq. (2)), browser- and cloud-based voice interaction, a Piston-backed coding sandbox, and a real focus-tracking anti-cheating mechanism. The same review found that the platform's webcam-based gaze and stress indicators are a client-side simulation with no underlying analysis pipeline (Section VI-E), and identified, as new findings specific to this revision, an untested failover module and two core AI-service modules with no automated test coverage at all (Section VI-B), a frontend dependency tree with two fixable moderate-severity advisories (Section VIII-A), and an unenforced timeout on the Gemini fallback path (Section III). We view the combination of a corrected implementation description, a formalized (and honestly scoped) adaptive mechanism, a small set of executed preliminary findings, and a concrete, not-yet-executed experimental protocol as a more useful contribution than a more polished description that could not be defended against the source code, and we identify executing that protocol as the immediate next step for this work.

References

[1] M. I. Tanveer, D. Gildea, and M. E. Hoque, "Automated analysis and prediction of job interview performance," *IEEE Transactions on Affective Computing*, vol. 9, no. 2, 2018.

[2] L. Chen, R. Zhao, C. W. Leong, B. Lehman, G. Feng, and M. E. Hoque, "Automated video interview judgment on a large-sized corpus collected online," in *Proc. IEEE Int. Conf. Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*, 2017.
[3] L. K. Joshila Grace et al., "Automated interview through online video interface," in *Proc. IEEE Conference*, 2023.
[4] A. A. J. Karim, S. Hoque, M. G. R. Alam, and M. Z. Uddin, "Automated personnel selection for software engineers using LLM-based profile evaluation," arXiv:2410.23365, 2024.
[5] S. Hoque, A. A. J. Karim, M. G. R. Alam, and N. Gope, "When LLM meets Fuzzy-TOPSIS for personnel selection through automated profile analysis," arXiv:2601.22433, 2026.
[6] A. Fabris, N. Baranowska, M. J. Dennis, D. Graus, P. Hacker, J. Saldivar, F. Zuiderveen Borgesius, and A. J. Biega, "Fairness and bias in algorithmic hiring: A multidisciplinary survey," *ACM Transactions on Intelligent Systems and Technology*, 2024.
[7] L. Zheng et al., "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
[8] A. Szymanski, N. Ziems, H. A. Eicher-Miller, T. J.-J. Li, M. Jiang, and R. A. Metoyer, "Limitations of the LLM-as-a-judge approach for evaluating LLM outputs in expert knowledge tasks," arXiv:2410.20266, 2024.
[9] S. Vatsal and H. Dubey, "A survey of prompt engineering methods in large language models for different NLP tasks," arXiv:2407.12994, 2024.
[10] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," arXiv:2212.04356, 2022.
[11] H. Misgna, B.-W. On, I. Lee, and G. S. Choi, "A survey on deep learning-based automated essay scoring and feedback generation," *Artificial Intelligence Review*, vol. 58, 2024.
[12] L. Labadze, M. Grigolia, and L. Machaidze, "Role of AI chatbots in education: Systematic literature review," *International Journal of Educational Technology in Higher Education*, vol. 20, no. 56, 2023.
[13] L. Huang et al., "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, 2024.
[14] J.-Y. Kuo, Z.-J. Wen, T.-F. Hsieh, and H.-X. Huang, "A study on the security of online judge system applied sandbox technology," *Electronics*, vol. 12, no. 14, p. 3018, 2023.
[15] S. Qian, K. Liu, P. C. Sruthi, L. Tan, and Y. Zhang, "Towards resiliency in large language model serving with KevlarFlow," arXiv:2601.22438, 2026.
[16] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020.
[17] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
[18] OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.