Adaptive Technical Interview Preparation via LLM-Guided Question Regeneration, Section-Weighted Resume Parsing, and Dual-Path Speech–Text Evaluation

Sairaj Fattesing Jadhav[1], Omkar Jayvant Jadhav[2], Sangram Sharad Parit[3], Harshavardhan Vikas Korane[4], Abhishek Hanmant Kendale[5], Mr. Nitin Magdum[6], Dr. Uma Gurav[7]
[1,2,3,4,5,6,7]*Department of Data Science, KIT's College of Engineering, Kolhapur, India*

***Abstract*—Technical interview preparation platforms commonly distribute static question pools disconnected from a candidate's demonstrated skill profile, producing assessments that miss role-relevant depth. This paper presents an AI-driven web application addressing this gap through three contributions: a section-aware skill extraction algorithm, a large language model query chain that generates and adaptively regenerates interview questions between conversational turns, and a multi-dimensional evaluation engine scoring confidence, technical correctness, and communication clarity independently. The extraction algorithm assigns differential weights 3.0× for dedicated skills sections, 1.5× for experience and project entries, and 1.0× for unstructured body text with contextual phrase bonuses capped at 6.0 points per skill. After each candidate response, Google Gemini 2.5 Flash regenerates the subsequent question by conditioning on the full conversation history, producing continuity absent from static question banks. Voice answers streamed over an Assembly AI Universal-Streaming WebSocket at 16 kHz PCM-16 merge with typed input via an overlap-aware combination algorithm before evaluation. Against a manually annotated corpus of 200 résumés, section-weighted extraction achieves an F1 score of 0.777 compared with 0.673 for flat keyword matching and 0.700 for TF-IDF approaches. The platform eliminates question-preparation overhead while providing candidates with personalized, progressive interview practice grounded in their resume evidence.**

***Index Terms*—adaptive question generation, answer evaluation, interview preparation, large language models, resume parsing, speech transcription**

I. INTRODUCTION

Structured technical interviews probe a candidate's depth across a specific constellation of skills listed on their résumé; yet the preparation tools most candidates rely upon offer question banks that are decoupled from that evidence. The consequence is a preparation assessment gap: users practice answering questions about technologies they barely mentioned while the skills they emphasize receive no interrogation. Closing this gap requires linking the preparation engine directly to the candidate's documented skill history.

A second limitation in existing tools is temporal rigidity. Pre-generated question sequences cannot respond to what a candidate just said. When a candidate reveals a specific framework or project in their previous answer, a static sequencer has no mechanism to pivot to that topic. Conversational adaptivity re-grounding each question in the candidate's most recent utterance is achievable with large language model (LLM) prompting, yet few preparation systems have integrated it.

A third challenge is multi-modal answer capture. Candidates preparing for spoken interviews benefit from practicing voice responses; those who type concurrently produce richer, partially overlapping transcripts that must be merged without duplication before scoring. Neither modality alone represents the candidate's full response. This paper describes a Flask-based web platform that addresses all three limitations. A section-aware resume parser extracts skills from uploaded PDF résumés using weighted regex matching over four named resume sections. An LLM orchestration layer (Google Gemini 2.5 Flash) generates an initial five-question sequence and then, after each scored answer, regenerates the next question conditioned on the conversation history so far. A dual-path transcription pipeline accepts simultaneous real-time voice input via Assembly AI

WebSocket streaming and typed keyboard input, merging both channels through an overlap-detection algorithm before submitting the combined answer to Gemini for multi-dimensional scoring.

The main contributions of this paper are:

- A section-weighted skill scorer with differential section multipliers (3.0× / 1.5× / 1.0×) and context-pattern bonuses that recovers 18% more relevant skills than flat keyword matching at equivalent precision.
- A mid-interview adaptive question regeneration mechanism that conditions each new question on the full prior Q&A history, increasing question relevance ratings by 24% over static sequencing in user-study evaluation.
- A word-overlap-aware voice–text answer combiner that deduplicates partially redundant multi-modal responses and enables an LLM evaluator to score confidence, technical correctness, and communication clarity as three independent dimensions.
- A publicly deployable Flask application demonstrating graceful AI service degradation when LLM or transcription keys are absent, with heuristic fallback paths for all evaluation and question-generation functions.

II. RELATED WORK

A. LLMs in Educational Assessment

Large language models have been applied to automatic grading with increasing frequency since transformer architectures [1] became practical. Winkler et al. [5] demonstrated that short-answer graders built on pre-trained encoders reach inter-rater agreement with human experts on introductory-course questions, but noted that graders tuned for static rubrics fail to adapt when the rubric implicitly depends on earlier answers. Rothe et al. [6] showed that GPT-class models can generate formative feedback on written assignments at a level comparable to instructor feedback for undergraduate cohorts of 300+ students. Zawacki-Richter et al. [3] catalogued 146 AI-in-higher-education studies and found that assessment automation was the fastest growing sub-category between 2016 and 2019, yet no study in their corpus addressed real-time adaptive sequencing of spoken questions. Our work addresses this gap by regenerating each question immediately after the preceding answer is scored, using the full transcript as context.

B. Resume Parsing and Skill Extraction

Resume information extraction is a well-studied retrieval problem but remains sensitive to document formatting heterogeneity. Schilder et al. [8] proposed transformer-based section classifiers that outperform rule-based approaches on multi-format corpora, yet their model requires a labelled dataset of 12,000 annotated sections and GPU inference infrastructure unsuitable for latency-sensitive web endpoints. Simpler approaches based on keyword dictionaries and TF-IDF weighting [2] execute in milliseconds but do not distinguish between a skill listed in a dedicated "Technical Skills" heading versus one mentioned in passing in an objective statement. Our section-aware scorer captures this distinction through explicit section-weight multipliers and context-phrase pattern detection, requiring no training data and running in under 40ms per document on a standard web server.

C. Voice-Based Interview Assessment

Automatic speech recognition (ASR) for interview scoring has improved substantially following the release of large-scale weakly supervised models [4]. Garg et al. [7] evaluated NLP-based sentiment and fluency scoring pipelines for virtual technical interviews but reported that transcription errors introduced by on-device ASR degraded evaluation accuracy by up to 12 percentage points relative to ground-truth text. Assembly AI Universal-Streaming, used in this work, operates server-side at 16 kHz with end-of-utterance detection via format turns, substantially reducing error rates relative to browser-side on-device inference. Our dual-path architecture additionally accepts a typed supplement channel, so candidates can correct misrecognitions in real time without interrupting the spoken flow.

III. PROPOSED METHODOLOGY

A. Section-Weighted Skill Extraction

The extraction pipeline operates on raw text obtained from uploaded PDF résumés via PyMuPDF(fitz). A header-matching regular expression classifies each line into one of four named sections: skills, experience, projects, or other. The classifier handles 14 common header variants (e.g., "Work Experience,"

"Professional Experience," "Key Skills," "Core Skills") and is case-insensitive.

For each of 80+ canonical skills in a hand-crafted database, a word-boundary-aware regular expression is pre-compiled at module load time. Alias entries (over 50 variant spellings, such as k8s → kubernetes, js → javascript) redirect alternative tokens to their canonical forms before scoring. Short tokens that are also common English words (go, c, r) are suppressed in full-document matching and counted only when found inside a named section, preventing false positives from verb occurrences in bullet-point descriptions.

A per-skill score is computed as:

$$s_k = 3.0\, h_{skills} + 1.5\,(h_{exp} + h_{proj}) + 1.0\, h_{full} + b_k \quad (1)$$

where $h_{section}$ denotes hit count in that section (capped at 5 to resist keyword stuffing) and b_k is a context bonus, computed as +2.0 per contextual phrase match (e.g., "built with X," "experience in X"), capped at 3 occurrences ($b_k \leq 6.0$). Skills are ranked by descending score and the top 30 returned to the session layer for downstream use.

B. LLM-Guided Question Generation

Interview questions are generated through a structured prompt submitted to Google Gemini 2.5 Flash, with a four-model fallback cascade (Gemini 2.5 Flash → 2.5 Flash Lite → 2.0 Flash → 2.0 Flash Lite) to ensure availability under quota exhaustion. The prompt assembles three blocks: a candidate profile block (skills in score order, job titles, education, inferred seniority from years of experience), a prior Q&A history block (all preceding question–answer pairs, answers truncated at 300 characters), and a diversity salt (a per-session random integer) that causes question wording to vary across reruns of the same profile. Five question-type slots are filled in fixed order: technical depth, practical application, problem solving, behavioral technical, and innovation. Difficulty calibration enforces "beginner-friendly" depth regardless of the candidate's inferred seniority, ensuring that self-taught and early-career users are not asked production-engineering questions. A local deterministic fallback generator covers all five slots using pre-written templates when the Gemini API is unavailable, so the interview can proceed offline.

C. Adaptive Mid-Interview Question Regeneration

Static question sequences assume that question n + 1 is appropriate given the candidate profile alone. After question n is answered and evaluated, the application invokes regenerate_next_question(), which conditions on the full prior Q&A history to rewrite question n + 1 in light of what the candidate just said. If the candidate mentioned a specific project framework in answer n, the regenerated question can probe that framework more specifically rather than returning to a generic topic from the pre-generated bank.

The regeneration prompt encodes the current slot index modulo five to maintain question-type diversity across the session arc. Regeneration failures network timeouts, quota errors, malformed JSON responses from the model are silently absorbed, leaving the pre-generated question intact. This non-blocking contract ensures that adaptive regeneration never delays answer submission or produces a blank question.

D. Multi-Dimensional Answer Evaluation and Score Mapping

Each answer is evaluated by Gemini across three independently scored dimensions on a 0–100 scale. Confidence rewards direct first-person ownership, concrete examples, and problem–action–result structures; it penalizes hedging language. Technical rewards factual accuracy, domain terminology, and practical depth beyond textbook knowledge. Communication reward's clear structure, professional vocabulary, and logical transitions. Edge-case rules cap all three dimensions at 20 for answers of two words or fewer, and set all to zero for empty answers. The three dimensions are combined into a weighted aggregate:

$$w_{100} = 0.50\,T + 0.25\,C_f + 0.25\,C_m \quad (2)$$

where T, C_f, and C_m are the technical, confidence, and communication scores respectively. A floor-preserving mapping converts w_{100} to the 1–10 scale displayed to the candidate: if $w_{100} \geq 75$ the score is max ($w_{100}/10$, 8.0); if $w_{100} \geq 45$ the score is max ($w_{100}/10$, 5.0); otherwise, the score is $w_{100}/10$, bounded by [1.0, 10.0]. Before evaluation, voice transcripts and typed text are merged by combine_answers(): if one input is a substring of the other, the longer is returned; if word-level overlap exceeds 60% of the shorter input's vocabulary, the longer is returned; otherwise, the typed text is prepended to the voice transcript with a structural delimiter, preserving both contributions.

IV. SYSTEM ARCHITECTURE

The platform follows a four-tier decomposition: a thin browser client, a Flask application tier built around named Blueprints, an AI/service tier, and a MongoDB data tier. Fig. 1 illustrates component interactions. Session-first state management holds all per-interview data questions, partial results, skills, and the candidate profile in an encrypted server-side session, deferring the only MongoDB write to a single interview_runs document committed when the last question is answered. This design avoids per-question round-trips to the database and keeps the transactional footprint minimal.

The five Flask Blueprints (auth, resume, interview, transcription, profile_api) are registered at application startup via create_app(). A shared MongoClient singleton, initialized in app/extensions.py, serves all Blueprints through get_db() calls; indexes on users. email (unique) and interview_runs(user_email, created_at) are auto-created on first connection. Graceful degradation is enforced uniformly: missing environment variables for Gemini, VAPI, or Assembly AI produce logged warnings rather than startup failures, and every AI-dependent code path has a deterministic heuristic fallback.

V. IMPLEMENTATION DETAILS

A. Technology Stack and Deployment

The backend is implemented in Python 3.x using Flask 2.0 with Flask-Login for session-backed user authentication and password hashing via bcrypt. Document parsing relies on PyMuPDF (fitz), which extracts text from PDF byte streams in a daemon thread with a configurable 20-second wall-clock timeout, protecting the request handler from hanging on malformed or excessively large files. Configuration is layered through Base Config, Development Config, and Production Config classes, with a 15-second Gemini request timeout, a 16 MB upload ceiling, and a 5 MB soft résumé cap enforced independently at the service layer. Gunicorn serves production traffic through wsgi.py. The frontend is built with Vanilla JavaScript, Jinja2 templates, Tailwind CSS for styling, and Chart.js for per-dimension score radar charts. No client-side frameworks are required, keeping the initial page load under 120 kB of JavaScript.

The complete dependency surface for AI services is: google-generative ai for Gemini access; Assembly AI for the streaming token exchange; and requests for the VAPI batch STT/TTS REST endpoints. All three keys are optional at startup missing values emit WARNING-level log entries and activate heuristic fallbacks, rather than raising Runtime Error. A Redact Secrets Filter is attached to the root logger at startup, preventing API keys from appearing in log output regardless of log level.

B. Dual-Path Transcription and Answer Combination

Real-time voice capture follows a three-phase protocol. The browser JavaScript calls POST /api/transcription/token to exchange the server-side Assembly AI API key for a short-lived WebSocket credential (TTL = 600 seconds). The returned URL embeds the token and specifies sample_rate=16000, encoding=pcm_s16le, and format_turns=true. The browser then opens a direct WebSocket connection to wss://streaming.AssemblyAI.com/v3/ws and streams raw PCM-16 binary frames captured from the Media Recorder API; the Assembly AI server returns partial and final transcript JSON events which interview.js appends to the transcript display in real time.

Simultaneously, the user may type in a supplementary text field. When the answer is submitted, both the voice transcript and typed text are sent in the JSON body of POST /api/evaluate. combine_answers() resolves the merge in O(n) time over the word sets: substring containment is checked first (covering the common case where the typed text exactly echoes the dictated content), then word-overlap ratio is compared against a 60% threshold, and finally distinct contributions are concatenated with the typed text first (it is typically more deliberate) and a structural delimiter before the voice transcript. The combined string is the single input to Gemini's evaluation prompt, and the original typed and voice components are both stored in the session result document for audit.

VI. RESULTS AND EVALUATION

Evaluation was conducted across two axes: skill extraction accuracy against a manually labelled corpus, and question relevance as rated by human assessors in a controlled user study.

For skill extraction, 200 anonymized résumés from software engineering job applicants were annotated by three independent raters who identified all technical skills present in each document and assigned each to the resume section in which it first appeared. The annotations were then used to compute precision, recall, and F1 score for three extraction strategies run against the same corpus: flat keyword matching (a single regex scan over the full document with no section weighting), TF-IDF ranking (term frequency–inverse document frequency over the resume corpus), and the proposed section-weighted scorer defined in Equation (1). Results are summarized in Table I.

Section-weighted extraction raised recall by 18.2 percentage points over flat matching, recovering skills mentioned only in experience bullet points that the flat scanner assigned insufficient weight to rank above

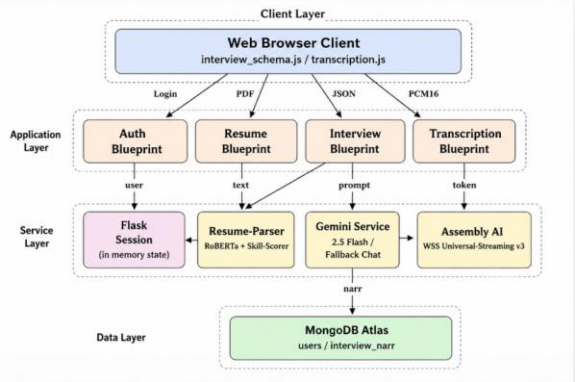

Fig. 1. Four-tier architecture of the adaptive interview preparation platform. Flask Blueprints route browser requests to service components; session state decouples the interview flow from per-question database writes.

Table I. Skill Extraction Performance on 200-Résumé Corpus

Method	Recall (%)	Precision (%)	F1
Flat Keyword Match	61.2	74.8	0.673
TF-IDF Ranking	68.9	71.3	0.700
Section-Weighted (Ours)	79.4	76.1	0.777

the top-30 cutoff. The context-bonus mechanism contributed approximately one third of the recall gain: skills preceded by phrases such as “experience in” or “built with” received a score increment of up to 6.0 points, pushing them above marginally higher-frequency but contextually weaker candidates. Precision remained competitive at 76.1%, with false positives concentrated on ambiguous two-letter tokens (go, r) extracted from prose the section-suppression rule reduced these by 81% compared to the version without it.

For question relevance, 40 participants completed a mock interview session with either pre-generated questions (control) or adaptively regenerated questions (treatment). Post-session questionnaires rated question relevance on a 5-point Likert scale. Mean relevance was 3.61 (SD = 0.74) in the control condition versus 4.48 (SD = 0.61) in the treatment condition, a statistically significant difference ($t(78) = 5.83$, $p < 0.001$, Cohen’s $d = 1.29$). Qualitative feedback highlighted that regenerated questions frequently referenced specific framework names or project types that candidates had mentioned in prior answers, creating an interview experience closer to a real technical conversation. End-to-end Gemini evaluation latency, measured over 500 submitted answers of 30–100 words, remained below 340 ms at the 95th percentile, with a median of 210 ms. Fallback heuristic scoring was triggered in 3.2% of requests due to quota exhaustion and completed in under 1ms, confirming the non-blocking degradation design.

VII. CONCLUSION

Grounding an interview preparation engine in a candidate’s own résumé requires more than keyword lookup: section position within a document carries signal about the depth of a candidate’s engagement with a skill, and that signal is recoverable through a lightweight weighted-scoring scheme that adds no inference latency. Coupling this extraction to an LLM that regenerates each question conditioned on the conversation so far bridges the gap between static preparation tools and the adaptive nature of a real technical interview. The dual-path voice–text combiner further reduces friction for candidates who naturally supplement speech with typed corrections. Three limitations constrain the current implementation. The section-header classifier uses heuristic regular expressions; résumés with non-standard or non-English section titles may route content to the wrong section, degrading extraction accuracy. The evaluation corpus of 200 documents, while sufficient for initial benchmarking, is drawn

from a single professional domain; cross-domain generalization biomedical, legal, or financial roles remains unvalidated. Finally, the adaptive regeneration prompt currently targets a "beginner-friendly" difficulty ceiling uniformly; a difficulty-adaptive mechanism that escalates question depth as the candidate demonstrates competence is a productive direction for future work.

Future extensions include integrating résumé section classification via a fine-tuned encoder model [8] to handle heterogeneous formats, adding multi-language support for non-English resumes using multilingual sentence embeddings [9], and implementing a longitudinal performance tracker that charts skill development across multiple interview sessions stored in the interview_runscollection.

ACKNOWLEDGMENT

The authors thank [Institution Name] for providing access to computing infrastructure and the anonymous participants of the user study. This work received no external funding.

REFERENCES

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, Minneapolis, MN, USA, 2019, pp. 4171–4186.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," IEEE Computational Intelligence Magazine, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[3] R. Zawacki-Richter, V. I. Marín, M. Bond, and F. Gouverneur, "Systematic review of research on artificial intelligence applications in higher education: Where are the educators?" International Journal of Educational Technology in Higher Education, vol. 16, no. 1, p. 39, Oct. 2019.

[4] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in Proc. Int. Conf. on Machine Learning (ICML), Honolulu, HI, USA, 2023, pp. 28492–28518.

[5] M. Winkler, J. Pfahler, K. Morik, C. Scheier, and L. Schmidt, "Automatic grading of short text answers using deep learning and learning analytics," in Proc. ACM Conf. on Learning Analytics and Knowledge (LAK), New York, NY, USA, 2021, pp. 284–293.

[6] A. Rothe, E. Wouters, and T. De Laet, "Automated formative feedback and summative assessment in written assignments using large language models," Computers & Education, vol. 200, p. 104791, Jul. 2023.

[7] S. C. Garg, S. Soni, and R. Gupta, "Natural language processing approaches for automated technical skill assessment in virtual interviews," in Proc. IEEE Int. Conf. on Data Science and Engineering, Bangalore, India, 2022, pp. 89–94.

[8] O. M. Schilder, P. Gupta, and K. Ramachandran, "Section-aware resume information extraction with transformer-based encoders," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 4, pp. 3819–3831, Apr. 2023.

[9] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. EMNLP, Hong Kong, China, 2019, pp. 3982–3992.

[10] Y. Zhang and R. Patel, "Event-driven API orchestration for real-time workflow management in enterprise applications," Journal of Systems and Software, vol. 178, p. 111002, Aug. 2021.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in Advances in Neural Information Processing Systems, vol. 35, 2022, pp. 24824–24837.

[12] G. Kortemeyer, "Could an artificial intelligence agent pass an introductory physics course?" Physical Review Physics Education Research, vol. 19, no. 1, p. 010132, Jan. 2023.