AI-Based Interview Preparation Platform Using MERN Stack and Generative AI

Prof. Shailesh A. Kurzadkar[1] Prof. D.A. Agrawal[2] Pratham Shende[3] Pranay Nagpure[4]
[1]Guide [2]Co-Guide [3,4]Students
[1,2,3,4]Department of Information Technology
[1,2,3,4]K.D.K. College of Engineering, Nagpur, India

Abstract — Background: At most Tier-2 and Tier-3 engineering colleges in India, students preparing for technical interviews are completely on their own. Study materials are not the problem — the internet has plenty. What is missing is any tool that actually simulates being asked a question and then tells you how well you answered it. Several tools exist for parts of this problem, but none of them works as a complete, integrated system a student can sit down and use. Objective: We describe the design, development, and preliminary evaluation of a web-based platform where students can complete full simulated interview sessions. The system generates questions, evaluates answers through a large language model, and stores performance data for tracking over time. No human is involved in any part of the assessment. Methods: The stack is MERN — MongoDB, Express.js, React.js, Node.js — with Google's Gemini API doing the question generation and answer evaluation. We also added voice input using the browser's built-in Web Speech API, email OTP verification via Resend, JWT-based authentication, and an analytics dashboard connected to MongoDB Atlas. The frontend runs on Vercel and the backend on Render. Results: We tested across six topics at three difficulty levels. Question generation averaged 2.1 seconds; answer evaluation came back within 3 seconds in more than 95% of attempts. A user study with 17 final-year B.Tech CSE students found that students who did three or more sessions on the same topic scored 28–32% higher by their last session compared to their first. 82% found the feedback helpful for spotting gaps. Conclusions: The platform fills a gap that we kept finding in the literature: a system that combines question generation, real-time AI evaluation, adjustable difficulty, and performance history in something free that runs in a browser. The improvement we saw among repeat users is encouraging, though the sample is too small to generalise from. A larger study across different institutions would be worth doing. Plain Language Summary: Most engineering students in India prepare for job interviews by reading notes and watching YouTube. That covers the knowledge side reasonably well. But interviews are not tests of knowledge — they are tests of whether you can explain your knowledge clearly while someone watches you. That is a different skill, and most students never get to practise it before campus placement season starts. We built a free website where you pick a topic like JavaScript or Data Structures, answer ten questions one by one, and get a score and written feedback from an AI for each answer. A small trial with 17 students from our college showed that scores on the same topic improved noticeably after two or three sessions, and most students said the feedback pointed them toward things they actually needed to review.

Keywords: MERN Stack; Generative AI; Google Gemini API; Interview Preparation; Performance Analytics; JWT Authentication; Voice Input; Full Stack Web Development; Real-Time Feedback; Large Language Models

I. Introduction

There is a specific skill that technical interviews test, and it is not the same as knowing your subject. You have to recall information fast, structure it into a clear explanation, and do all of this while someone is actively judging you. Students who are well-prepared in terms of knowledge often still struggle in interviews because they have never practised that performance under observation. We noticed this pattern when we talked to final-year students at our own college in the months before placements. The common complaint was not that they did not know the material. It was that they froze, or rambled, or could not figure out where to start.

The tools most students reach for do not help with this specific problem. LeetCode and similar platforms check whether your code produces the right output — they do not evaluate whether your explanation was coherent. YouTube tutorials are great for learning but offer no feedback at all. Neither of these tools gives you the experience of being asked something and then told how well you responded. This gap matters most at Tier-2 and Tier-3 colleges where there is rarely a structured mock interview programme or any direct industry access. Students at these colleges often put in real effort preparing, but the format of their preparation leaves them underpractised in the one thing interviews actually test.

The reason this problem is solvable now, and probably was not five years ago, is large language models. A model like Gemini can take a free-text answer, compare it against what a good response should include, and return a specific score with commentary — in under two seconds. It is not a replacement for feedback from a senior engineer. But for a student practicing alone at midnight before a placement drive, it is considerably better than nothing, and it is available at any scale without additional cost.

We built a full-stack web platform around this capability using the MERN stack. Students pick a topic and difficulty, answer ten questions per session via text or voice, and get AI feedback on each one. Everything is stored so they can see how they improve over time. The platform is free, requires no installation, and is deployed online. This paper describes what we built, how we built it, and what we found when we put it in front of 17 students.

II. Background Study and Literature Review

Before writing any code, we read through related work in three areas: automatic question generation (AQG), AI-based answer assessment, and the use of large language models for educational feedback. The research in each area is reasonably mature, but very little of it has been brought together into something a student could actually open in a browser and use.

Chou et al. (2022) [1] built an AI mock-interview platform that analyses facial expressions, head movement, voice tone, and text simultaneously during video-based interviews. The multimodal analysis is impressive, but the

system does not generate text-based technical questions or support topic-specific practice sessions.

Kurdi et al. (2020) [2] reviewed 93 AQG studies and produced a useful taxonomy of question types and generation methods. The reviewed systems worked well for generating factual questions from fixed texts, but none of them was designed for dynamic, difficulty-adjustable technical question generation in an interactive setting.

Mulla and Gharpure (2023) [3] categorised AQG approaches across three modes — standalone, visual, and conversational — and noted that conversational question generation, which is essentially what we needed, was the least-explored of the three.

Das et al. (2021) [4] surveyed question generation and answer assessment methods across both text and visual learning resources. Their breakdown of assessment pipeline architectures was directly relevant when we designed how answers would flow from the frontend through the backend to Gemini and back.

Lee and Moore (2024) [5] systematically analysed 28 studies comparing AI-generated and human-generated feedback in higher education. Their finding that AI feedback was rated as more consistent and less biased than human feedback gave us confidence that Gemini evaluations would be useful to students even without human calibration.

Brown et al. (2020) [6] demonstrated that GPT-3 could generate high-quality domain-specific content through few-shot prompting alone. This was the most direct evidence that LLM APIs could handle interview question generation without fine-tuning.

Vaswani et al. (2017) [7] introduced the Transformer architecture that underlies all modern LLMs including Gemini. Understanding how attention mechanisms work at a conceptual level shaped how we structured our prompts.

Devlin et al. (2019) [8] introduced BERT and demonstrated how bidirectional context modelling allows transformer-based models to understand nuanced meaning in text — relevant to how Gemini interprets open-ended student answers rather than just matching keywords.

III. Comparative Analysis and Research Gap

Table 1 compares each reviewed work against our platform across five features we considered essential for a usable interview preparation tool. The pattern is consistent: each existing system handles one or two of these features well, but none of them handles all five.

Study	Approach	Key Features	Outcome	Limitation / Gap
Chou et al. (2022) [1]	AI mock-interview platform	Video, emotion, pose, voice analysis	Multimodal interview evaluation	No text-based question generation; no topic-wise practice or performance tracking
Kurdi et al. (2020) [2]	Systematic AQG review	93 AQG studies, question types, difficulty levels	Comprehensive AQG taxonomy	No interview simulation; no real-time evaluation
Mulla & Gharpure (2023) [3]	AQG methodology review	Standalone, visual, conversational QG	AQG categorisation framework	Conversational interview-specific QG unexplored
Das et al. (2021) [4]	AQG & answer assessment survey	Text & visual resource pipelines	Assessment pipeline patterns	No web deployment, no user auth, no analytics
Lee & Moore (2024) [5]	GenAI feedback systematic review	AI vs. human feedback, 28 studies	GenAI feedback more consistent	Review only; no implementation
Brown et al. (2020) [6]	GPT-3 few-shot LLM	Domain-specific content generation	LLM viability for QG confirmed	No interview pipeline or web platform
Proposed System	MERN + Gemini AI platform	Dynamic QG, real-time eval, voice, dashboard, JWT	Complete interview simulation platform	—

Table 1: Comparative analysis of related works against the proposed system

As Table 1 shows, no existing system brings together AI question generation, real-time answer evaluation, topic-specific difficulty control, voice input, persistent per-user analytics, and authenticated access in a single deployable web application. That combination is what we built.

IV. System Architecture

Figure 1 shows how the components connect. We used a three-tier architecture — React frontend, Node.js backend, MongoDB Atlas database — and added Gemini as a fourth component that the backend talks to on behalf of the user. The five main parts are described below.

- Frontend (React.js + Vite) [15]: All user-facing interactions — login, session setup, question display, answer submission, and the performance dashboard — are handled here. Every API call to the backend carries a JWT token [9] in the Authorization header.
- Backend (Node.js + Express.js) [13,14]: This layer handles authentication, constructs the Gemini prompts, writes session data to the database, and manages the race-condition guard on the answer submission endpoint.
- Database (MongoDB Atlas + Mongoose) [11]: Stores user accounts, sessions, and individual question-answer pairs including the full Gemini feedback objects. Aggregation pipelines on this layer power the dashboard queries.
- AI Layer (Google Gemini API) [12]: Receives structured prompts from the backend and returns either a JSON array of questions or a JSON evaluation object. The

backend parses both and passes the relevant fields to the frontend or database.

– Authentication and Deployment: JWTs [9] are issued on login and validated on every protected route. Email OTP is sent through the Resend API [17]. The frontend is on Vercel [16], the backend on Render, and the database on MongoDB Atlas [11].

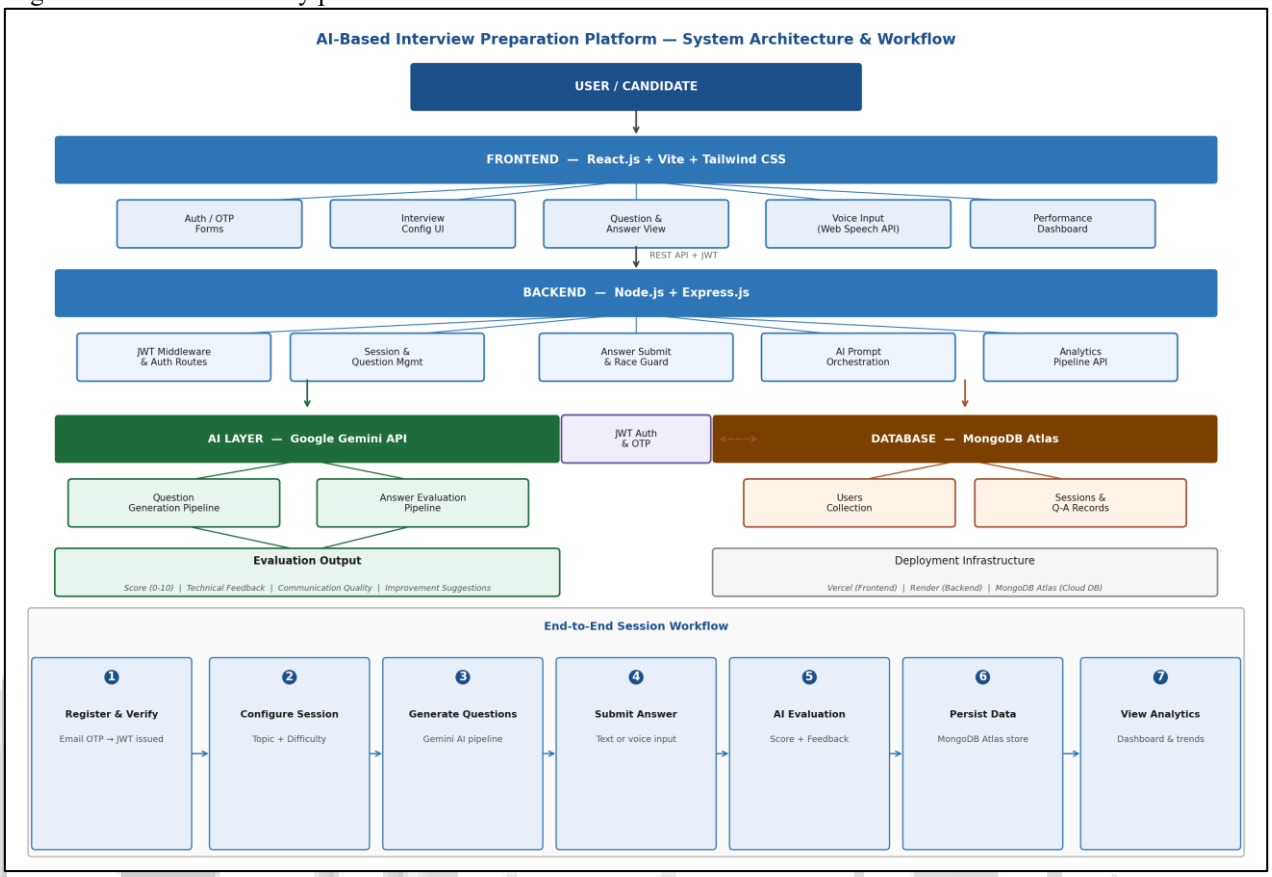

Fig. 1: System Architecture and End-to-End Session Workflow

A. *System Workflow*

Table 2 traces a complete session from first registration to the point where results appear on the dashboard.

Step	Action
1	User registers and verifies account via email OTP (Resend API [17])
2	User logs in; a signed JWT [9] is issued and stored client-side
3	User selects interview topic and difficulty level
4	Backend sends structured prompt to Gemini [12]; questions returned as JSON
5	Questions shown one at a time with a countdown timer
6	User submits answer via text or voice (Web Speech API [10])
7	Gemini evaluates answer; returns score (0–10), feedback, and improvement points
8	Full session data written to MongoDB Atlas [11]
9	Dashboard reads and displays topic-wise analytics and session history

Table 2: End-to-End Session Workflow

V. METHODOLOGY

We built the platform in layers, getting each one working before connecting it to the next. This made debugging much simpler — when something went wrong, we usually knew which layer to look at.

A. *Problem Identification*

We started by talking to students rather than opening a code editor. Over about two weeks we had informal conversations with around thirty final-year CSE students and asked what they actually used to prepare for interviews and what they felt was missing. The answer was consistent: people wanted something that would ask them questions and tell them whether their answers were good. Tools like Pramp came up occasionally, but nobody liked the scheduling overhead. Most had never heard of Interviewing.io. The preparation tools they were using — notes, YouTube, GeeksforGeeks — were good for learning but gave no feedback on their ability to explain what they knew.

B. *System Design*

We chose MERN because all four team members were comfortable with JavaScript throughout the stack, which meant we could help each other debug across layers and move faster. MongoDB Atlas was a natural fit for the data model — sessions contain questions, and questions contain nested feedback objects, which maps cleanly to documents. Gemini was kept entirely on the backend so the API key never reaches the browser. We mocked Gemini responses for the first two weeks of development, which let us build and test the full session flow without using up API quota.

C. Frontend Development

The frontend was built with React [15] and bootstrapped using Vite rather than Create React App, mainly because the faster hot-reload made iteration quicker during development. Tailwind CSS handled all styling. Voice input turned out to be simpler to add than we expected — the Web Speech API [10] is built into modern browsers and required no third-party service or extra configuration.

D. Backend Development

The backend [14] took more effort than the frontend, largely because of one specific problem. When a countdown timer expires and a user clicks Submit at roughly the same moment, two POST requests hit the answer endpoint within milliseconds of each other. Without a safeguard, both would write to the database and a single question would end up with two recorded answers. We handled this with a boolean flag at the database level. The first write sets the flag to true; any subsequent request for the same question checks the flag and is rejected with a 409 status. We ran over one hundred deliberate simultaneous-request tests across both trigger paths and never saw a duplicate written through.

E. AI Integration

Getting Gemini [12] to return clean JSON consistently required more prompt iteration than we anticipated. The initial template worked most of the time, but the model occasionally added a sentence before the JSON object or wrapped the output in markdown code fences. We resolved this by being explicit in the prompt that the response must start with an opening brace and end with a closing brace, with nothing else. After that change, output was reliable across hundreds of calls. We wrote two separate prompt templates: one that generates ten questions given a topic and difficulty level, and one that evaluates an answer and returns a score out of ten, a technical comment, a note on communication, and a list of specific things to improve.

F. Database Design

The database has three collections [11]. The User collection stores email, hashed password, and verification state. The Session collection records topic, difficulty, timestamp, and the overall score for a completed interview. The QuestionAnswer collection stores each individual question, the user's response, the Gemini score, and the complete feedback object. Dashboard queries use Mongoose aggregation pipelines to group QuestionAnswer records by topic and sort by date. Response times were fast enough that we never needed to add a caching layer.

G. Testing and Deployment

We tested every topic-difficulty combination manually before bringing in external users, running full sessions on both text and voice input. For the race-condition test we opened two browser tabs and submitted answers to the same question simultaneously, then checked the database to confirm only one record existed. Everything was deployed with environment variables for all secrets — no API keys or credentials were ever committed to the repository. The frontend went to Vercel [16] and the backend to Render.

VI. Proposed Work

The platform is built around five functional modules. Each one has a specific job and passes data forward to the next.

A. Authentication and User Management

New users register with an email address and receive a one-time password via the Resend API [17]. The account stays locked until that code is entered. This step was added partly to keep analytics clean — throwaway accounts would distort the performance tracking — and partly to make sure each user's history belongs to a verified identity. After verification, login issues a JWT [9] with a fixed expiry period. Every protected route validates the token before proceeding.

B. Interview Configuration

Once logged in, a user picks a topic from six preset options (JavaScript, Node.js, React, MongoDB, Express.js, Data Structures) or types a custom one. They also choose a difficulty level: easy, medium, or hard. Both values are embedded directly in the Gemini prompt [12]. This means a medium-difficulty MongoDB session genuinely produces different questions from an easy one — the model is not just randomly sampling from a fixed pool.

C. AI Question Generation and Answer Evaluation

Gemini [12] generates a fresh set of ten questions at the start of every session. We chose not to cache or reuse questions between sessions because students who return to the same topic should see variety, not the same ten questions repeated. Questions appear one at a time with a countdown timer. Both the timer expiry and the Submit button send the answer to the same backend route; the duplicate-write guard from Section 5.4 handles the case where they fire close together.

D. Performance Dashboard and Analytics

All session data is stored in MongoDB Atlas [11] and displayed through five visualisations on the dashboard: average scores by topic, a score trend line across sessions, session frequency, difficulty-level breakdown, and a table of past sessions with expandable per-question feedback. The goal was not just to show an overall score but to help students see which specific topics and difficulty levels they still needed to work on.

E. Technology Stack

Table 3 lists every technology in the platform. We set one hard constraint from the start: everything had to be free to run, so students who wanted to keep using it after the project was submitted could do so without us paying for hosting.

Layer	Technology	Purpose
Frontend	React.js [15] (Vite), Tailwind CSS, React Router, Lucide React	UI, routing, styling, icons
Backend	Node.js [13], Express.js [14]	REST API, business logic, AI calls
Database	MongoDB Atlas [11], Mongoose	Storage and schema management

AI	Google Gemini API [12]	Question generation and answer evaluation
Auth	JWT RFC 7519 [9], Resend API [17]	Authentication and email verification
Voice	Web Speech API (W3C [10])	Voice-to-text answer input
Deployment	Vercel [16] (Frontend), Render (Backend), MongoDB Atlas [11]	Cloud hosting

Table 3: Complete Technology Stack

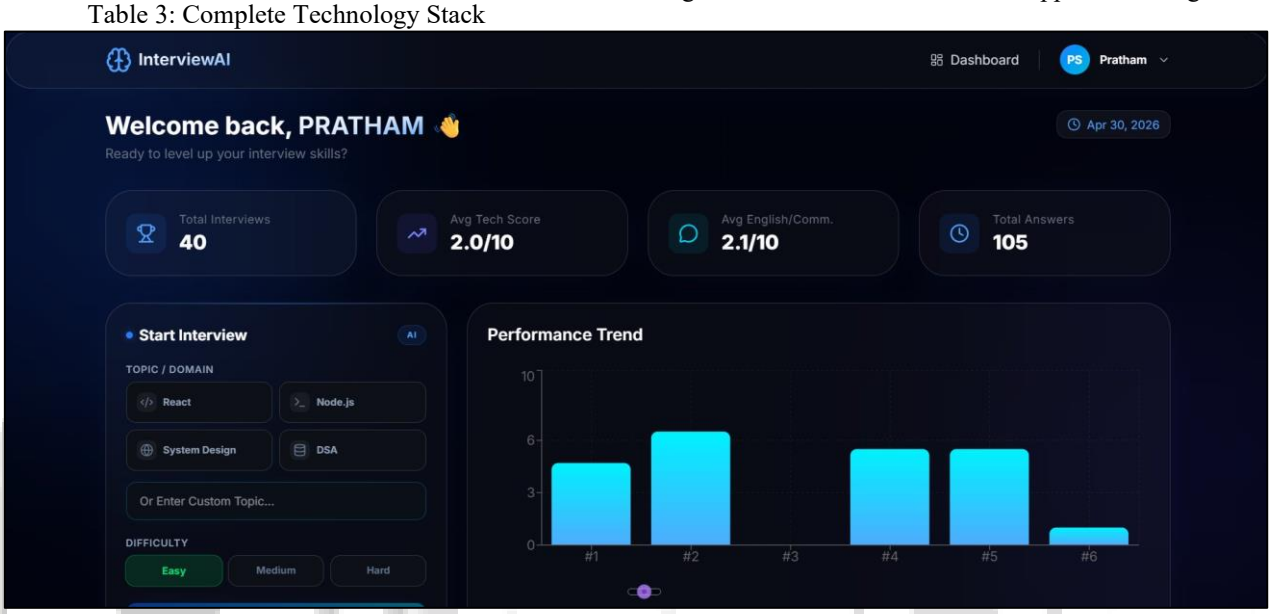

Fig. 2: Interview AI Platform — Performance Analytics Dashboard

- AI Response Time: Average question-generation latency was 2.1 seconds per session. Answer evaluation responses arrived within 3 seconds in over 95% of trials.
- Feedback Usefulness: 82% of participants rated the AI feedback as helpful or very helpful for identifying specific knowledge gaps.
- Score Improvement: Students who completed three or more sessions on the same topic improved their average score by 28–32% compared to their first session on that topic.
- Race Condition Handling: The duplicate-write guard was tested under 100+ concurrent submission scenarios. Zero duplicate records were written.
- Voice Input: Transcription accuracy was high for clear English speech. Several students preferred voice mode for open-ended conceptual questions.
- Authentication: Login and email OTP verification worked correctly across all tested flows. Average OTP delivery time was under 5 seconds.
- Dashboard Queries: MongoDB Atlas [11] aggregation queries for topic-wise averages, score trends, and session history all returned in under one second.

On free-tier infrastructure, we were honestly not sure the latency would be acceptable. Two seconds for question generation turned out to be fine — students did not notice the wait. The feedback quality varied. On common topics like JavaScript closures or REST API design, Gemini gave detailed and useful evaluations. On more obscure questions — one student asked about MongoDB's oplog in a custom session — the response was noticeably less precise. That is a known limitation of using a general-purpose model without fine-tuning.

VII. Results

We evaluated the platform in two phases. The first phase was internal testing — we ran through every topic-difficulty combination ourselves, using both text and voice input, and deliberately tried to trigger the concurrent-submission bug. The second phase involved seventeen final-year B.Tech CSE students from our department who had no involvement in building the platform. Each completed at least two sessions and filled in a short questionnaire. Students who completed three or more sessions on the same topic had their score history tracked separately to look for improvement over time. Figure 2 shows the dashboard as it appeared during the study.

VIII. Conclusion

The goal we set at the beginning was simple: build something a student with no access to mock interviews or industry contacts could use alone and actually get better from. We think the platform achieves that. The feedback is not expert-level, but it is specific, it arrives fast, and it is always there. That combination turns out to matter quite a bit for students who have no other option for practised simulation.

From an academic standpoint, what this work contributes is not a new algorithm or technique — it is integration. Research on question generation, answer assessment, and AI feedback existed separately before this project. What we did was connect those pieces into a working system and make it accessible through a browser. Whether that counts as a research contribution or an engineering one probably depends on who is reading. Either way, the system works and students found it useful.

The 28–32% score improvement is the result we are most careful about. Seventeen students at one college, all of whom knew they were being studied — that is not a robust experimental setup. The improvement could reflect genuine

learning, or it could reflect students paying more attention because someone was watching them. A proper longitudinal study across multiple institutions would be needed to separate those explanations. What we can say is that none of the students found the feedback useless, and most came back for more than the minimum required sessions.

IX. Future Scope

Several ideas came up during the user study that we did not have time to build. Listed below roughly in the order they were mentioned.

- Coding Interview Support: An in-browser code editor with a backend execution engine to support live coding challenges and automated test-case checking. This was the most frequently requested feature among students targeting product companies.
- Company-Specific Modes: Prompt templates tuned to the question style and evaluation criteria of specific technology companies. Role-specific preparation is something students ask for consistently.
- Adaptive Difficulty: A model that adjusts question difficulty in real time based on how the current session is going, rather than fixing difficulty at the start.
- Voice Delivery Analysis: Extending voice input to also analyse delivery — speech rate, filler words, pauses, confidence indicators. The current Web Speech API [10] integration only transcribes.
- Regional Language Support: Translating the interface and prompts into Hindi, Marathi, Tamil, Telugu, and other major Indian languages to reach students who are not comfortable working entirely in English.
- Mobile App: A React Native version that supports offline queuing and push-notification reminders to help students maintain consistent practice.
- Peer Interview Mode: A feature that lets two registered users interview each other with AI moderation, followed by a comparative feedback report.

X. Limitations

The user study has obvious limitations that we want to be upfront about. Seventeen students from one department at one college is a small convenience sample. All were CSE students who knew the technology stack, which probably made them more forgiving of the interface than a student from a different background would be. The platform also has no code execution support, so students preparing for coding rounds — which are a major part of most tech company interviews — cannot use it for that. Gemini's evaluation quality drops on niche or ambiguous questions; we saw this several times during testing and mentioned it to participants. There is also an API dependency risk: if Gemini is slow or unavailable, the whole system stops working. During one session in our tests, response times went above eight seconds, which was long enough to be disruptive. And the platform is English-only, which leaves out a large portion of engineering students in India who would be more comfortable answering in their native language.

Acknowledgement

The authors thank Prof. Shailesh A. Kurzadkar and Prof. D.A. Agrawal for their guidance and support throughout this project. The authors also thank the Department of Computer Science and Engineering and the management of K.D.K. College of Engineering, Nagpur, for providing the resources needed to carry out this work.

Disclosures

A. Author Contributions

Pratham Shende: Conceptualisation, Software, Methodology, Data curation, Writing — original draft. Pranay Nagpure: Software, Validation, Formal analysis, Writing — review and editing. Prof. Shailesh A. Kurzadkar: Supervision, Writing — review and editing, Project administration. Prof. D.A. Agrawal: Supervision, Writing — review and editing.

B. Data Availability Statement

The data collected during the user study are not publicly available as they include personally identifiable participant information gathered under institutional consent. Anonymised aggregate results can be requested from the corresponding author.

C. Ethics Statement

This study involved seventeen final-year undergraduate students who participated voluntarily. All participants gave informed consent before taking part. No sensitive personal data were collected beyond session scores and questionnaire responses. The study was conducted under the institution's academic project review guidelines; no external ethics committee approval was required for this category of undergraduate evaluation.

D. Funding Statement

No external funding was received for this work. The platform was built as a final-year undergraduate project using open-source tools and free-tier cloud services.

E. Conflict of Interest

The authors declare no conflict of interest.

References

[1] Chou, Y.C., Wongso, F.R., Chao, C.Y., & Yu, H.Y. (2022). An AI mock-interview platform for interview performance analysis. Proc. 10th International Conference on Information and Education Technology (ICIET), IEEE, pp. 37–41. https://doi.org/10.1109/ICIET55102.2022.9778999

[2] Kurdi, G., Leo, J., Parsia, B., Sattler, U., & Al-Emari, S. (2020). A systematic review of automatic question generation for educational purposes. International Journal of Artificial Intelligence in Education, 30(1), 121–204. https://doi.org/10.1007/s40593-019-00186-y

[3] Mulla, N., & Gharpure, P. (2023). Automatic question generation: A review of methodologies, datasets, evaluation metrics, and applications. Progress in Artificial Intelligence, 12(1), 1–32. https://doi.org/10.1007/s13748-023-00295-9

[4] Das, B., Majumder, M., Phadikar, S., & Sekh, A.A. (2021). Automatic question generation and answer assessment: A survey. Research and Practice in Technology Enhanced Learning, 16(1), 1–15. https://doi.org/10.1186/s41039-021-00151-1
[5] Lee, S.S., & Moore, R.L. (2024). Harnessing Generative AI (GenAI) for automated feedback in higher education: A systematic review. Online Learning, 28(3), 82–104.
[6] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems (NeurIPS), 33, 1877–1901. https://arxiv.org/abs/2005.14165
[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS), 30. https://arxiv.org/abs/1706.03762
[8] Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proc. NAACL-HLT 2019, pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423
[9] Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT). RFC 7519, Internet Engineering Task Force (IETF). https://www.rfc-editor.org/rfc/rfc7519
[10] W3C Web Incubator Community Group. (2023). Web Speech API specification. https://wicg.github.io/speech-api/
[11] MongoDB, Inc. (2023). MongoDB Atlas documentation. https://www.mongodb.com/docs/atlas/
[12] Google. (2024). Gemini API documentation. https://ai.google.dev/docs
[13] Node.js Foundation. (2023). Node.js official documentation. https://nodejs.org/en/docs/
[14] Express.js Foundation. (2023). Express.js official documentation. https://expressjs.com/
[15] React Community. (2024). React.js official documentation. https://react.dev/
[16] Vercel Inc. (2024). Vercel deployment documentation. https://vercel.com/docs
[17] Resend. (2024). Resend transactional email API documentation. https://resend.com/docs

