iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80429

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Review on Adaptive Interview Prep with Generative AI and Analytic

Muskan Lodhi[1], Sharvari Kamble[2], Rahul Yadav[3], Pratik Tikare[4], Supriya Pawar[5]
[1, 2, 3, 4]*Students,* [5]*Assistant Professor, Department of Computer Engineering, Bharat College of Engineering, University of Mumbai, Maharashtra, India*

Abstract: ***Technical interview preparation remains a deeply fragmented challenge for engineering students and working professionals. Existing tools are either statically structured, heavily dependent on external AI APIs, or offer no personalised performance insight beyond a binary pass-fail outcome. This paper presents Crack-it, a full-stack web platform that fuses two fundamentally distinct assessment methodologies under one roof: a Generative AI pipeline for resume-driven conversational interviews and a Zero-API, deterministic local evaluation engine for domain-based screening. The platform analyses 350-plus curated industry-standard questions across seven technical domains and evaluates user responses through a weighted scoring model that combines technical keyword density at 70% and semantic Dice coefficient similarity at 30%, supplemented by real-time behavioural signals captured via an in-browser computer vision pipeline. An isolated server-side code execution sandbox enforces strict resource limits and pattern-based security guards for multi-language coding challenges. Firebase Firestore maintains a persistent, evolving user profile across sessions, and a multi-tier AI fallback architecture ensures service continuity for resume interviews even under API quota constraints. Evaluated across functional correctness, structural match, and communication quality, the platform demonstrates that rigorous interview preparation does not require a paid AI subscription for every interaction.***

Keywords: Generative AI, Technical Interview Preparation, Deterministic Evaluation, Adaptive Learning, Behavioural Analytics, Large Language Models.

I. INTRODUCTION

The hiring funnel for software engineering roles has, over the last decade, converged on a remarkably consistent structure: an initial technical screening, followed by one or more live coding rounds, and finally a behavioural or system-design discussion. Each of these stages demands a different kind of preparation, yet the tools available to candidates rarely address more than one stage at a time. Platforms like LeetCode and HackerRank handle algorithmic coding well but offer nothing for verbal technical questioning. Pramp provides peer-to-peer mock interviews but depends entirely on the availability and capability of other users. Static question banks from sites like GeeksforGeeks cover theory but cannot evaluate how fluently a candidate articulates a response or whether they maintain composure under simulated interview pressure.

The gap that motivated this work is not a shortage of practice content —there is arguably too much of it. The actual deficit is in personalisation and multi-modal feedback. A candidate who fluently explains recursion but fumbles under recorded-camera conditions does not have the same preparation need as one who is technically weak but communicates confidently. Treating both identically, as most tools do, wastes the candidate's most limited resource: time spent preparing.

Crack-it was designed around this observation. Rather than building another question bank, the system builds an evaluation framework that adapts to the candidate's demonstrated performance and measures multiple dimensions of readiness simultaneously. Domain-based interview questions are evaluated entirely without API calls using a deterministic, locally computed scoring model, a deliberate design choice that eliminates latency, reduces cost, and makes evaluation fully reproducible. Resume-driven interviews, by contrast, leverage Google Gemini's language model to generate contextually grounded questions tied to the specific projects and technologies a candidate has personally claimed. Coding challenges are executed in a sandboxed subprocess on the server with language-specific security guards, and a structural match engine scores the solution against a reference implementation without requiring the code to merely produce the correct output, it also verifies that the underlying algorithmic logic aligns with an optimal approach.

The platform currently supports seven technical domains — Web Development, Data Science, DevOps, Cybersecurity, Cloud Computing, Artificial Intelligence, and UI/UX Design — and thirty graded coding problems spanning beginner, medium, and advanced difficulty.

A skill gap analysis module maps a user's stated competencies against role-specific requirements and produces a phased learning roadmap. All session data is persisted in Firebase Firestore and surfaced through a progress analytics dashboard with trend charts and downloadable PDF reports.

II. MOTIVATION

Two distinct candidate failure modes motivated this work. The first is strong technical knowledge that cannot be articulated fluently under recorded-camera conditions, a problem no existing text-based platform can detect. The second is over-rehearsed generic answers that collapse when an interviewer probes the candidate's own stated resume experience, a gap that resume-agnostic tools cannot address. A third practical motivation was cost: routing every domain-question evaluation through an LLM API introduces latency above 1.5 seconds and makes free-tier operation economically unsustainable at scale, which the deterministic engine resolves by scoring in under 200 milliseconds with no network call.

III. PROBLEM STATEMENT

Technical interview preparation platforms in their current form suffer from four compounding deficiencies. First, they are domain-narrow: a platform that excels at algorithmic coding cannot evaluate whether a candidate can explain the difference between containerisation and virtualisation verbally, and vice versa. Candidates must navigate multiple disconnected tools, each with its own account, progress state, and feedback format, making it difficult to get a unified picture of overall readiness.

Second, existing platforms are evaluation-shallow. A binary correct-incorrect verdict on a coding problem, or a simple view count on a video explanation, does not constitute meaningful feedback. Candidates need to understand not just what they got wrong but why — whether the gap is in conceptual depth, in the ability to communicate under observation, or in the structural quality of their code versus an optimal solution.

Third, the assumption that real-time AI evaluation requires a live API call for every interaction creates both a latency bottleneck and a cost barrier. For domain-based interviews covering hundreds of technical questions across seven fields, this assumption makes automated practice economically unsustainable for any platform aiming at free or low-cost access.

Fourth, there is no existing open tool that simulates the multi-modal reality of a modern technical interview, where the candidate is simultaneously being assessed on what they say, how they say it, and whether they maintain focus and composure throughout. The absence of this multi-modal simulation means that even a candidate who has thoroughly prepared on existing platforms may encounter unfamiliar anxiety responses during the actual interview because they have never rehearsed under observed, recorded conditions.

The proposed system directly addresses each of these four gaps within a single, integrated platform.

IV. LITERATURE SURVEY

Sr. No	Author(s) & Year	Title	Key Findings	Limitations	Relevance to Proposed System
1	Zhang et al., 2019	Automated Verbal Interview Scoring Using NLP	Transformer embeddings can capture semantic relevance beyond keyword overlap in spoken interview responses.	Limited to text; no multi-modal or coding evaluation.	Validates the semantic similarity component of Crack-it_AI's answer scoring engine.
2	Corbett & Anderson, 1994	Knowledge Tracing for Adaptive Tutoring	Bayesian mastery estimation enables personalised difficulty progression in intelligent tutoring systems.	No application to live technical interviews; purely academic exercises.	Informs the difficulty-adjustment mechanism used between consecutive questions.
3	Naim et al., 2015	Automated Prediction of Interview Performance from Nonverbal Cues	Eye contact, speaking rate, and facial engagement are measurable predictors of interview outcome.	Requires specialised hardware; no browser-based deployment pathway.	Motivates the in-browser computer vision pipeline for behavioural metric capture.
4	Kasneci et	ChatGPT for Good?	LLMs can personalise	No implementation for	Supports the use of

Sr. No	Author(s) & Year	Title	Key Findings	Limitations	Relevance to Proposed System
	al., 2023	Opportunities and Challenges of LLMs in Education	assessment and generate context-relevant questions, but hallucination risk must be managed.	coding or multi-domain interview simulation.	Gemini for resume-driven interviews and structured JSON output schemas.
5	Belhumeur et al., 2020	Secure Sandboxed Code Execution for Online Judges	Subprocess isolation with resource limits and pattern-based injection guards is effective at 99.7% attack prevention.	High latency for compiled languages; no structural match scoring.	Directly informs the sandbox design in the coding module.
6	Campion et al., 1997	A Review of Structure in the Selection Interview	Structured interviews with consistent scoring criteria are substantially more predictive of job performance than unstructured ones.	Does not address automation or real-time feedback.	Provides the theoretical basis for consistent question banks and scoring rubrics per domain.
7	Kumar & Sharma, 2021	Adaptive E-Learning Systems: A Survey	Systems that adjust content difficulty based on learner performance improve knowledge retention by a measurable margin.	Focused on academic learning; not applied to interview simulation.	Supports the adaptive question selection logic across difficulty tiers.
8	OpenAI, 2023	GPT-4 Technical Report	Large language models with structured JSON output modes reduce hallucination risk in constrained evaluation tasks.	Proprietary; cost-prohibitive for every-interaction evaluation.	Motivates the hybrid design: LLM for resume interviews, deterministic engine for domain interviews.

Table 1: Literature Survey

The survey reveals that while adaptive learning, NLP-based interview scoring, and browser-based behavioural analysis have each been explored independently, no prior system unifies all three under a single, zero-subscription platform. The proposed system fills this gap by combining a deterministic local evaluation engine with selective generative AI usage, in-browser computer vision, and a secure code execution sandbox.

V. METHODOLOGY

The platform is built around two fundamentally distinct assessment methodologies, each applied where it is most appropriate, rather than forcing a single approach across all interaction types.

A. Generative AI Methodology (Resume-Centric Interviews)

This pathway employs Google Gemini's large language model to conduct personalised conversational interviews grounded in the candidate's own uploaded resume. The system extracts up to 10,000 characters from the PDF using pdfjs-dist and embeds this text as a context block in every prompt sent to Gemini's generateQuestion API. Questions are dynamically tailored to the specific projects, technologies, and job roles mentioned in the resume, meaning that two candidates with identical technical backgrounds but different experience descriptions will face different question sets. Each question generated includes a difficulty level, a topic classification, a reason field explaining why that question was selected given the candidate's history, and an expected answer for evaluation reference.

Evaluation is performed by a subsequent Gemini call to evaluateAnswer, which assesses the response along three axes: technical depth of explanation, consistency between the stated answer and the resume claims, and relevance to the specific project challenge being discussed. The evaluator returns a score from 0 to 10, a feedback string, and a next_difficulty recommendation.

B. Deterministic ML-Simulated Methodology (Domain-Based Interviews)

For the seven technical domain paths, every evaluation is computed locally with zero API calls. This Zero-AI approach relies on a curated knowledge base of 350-plus industry-standard questions distributed across seven domains and three difficulty tiers, each paired with a reference expected answer. When a candidate submits a response, the following pipeline executes entirely on the application server:

Technical Accuracy (70% weight): The user's transcript and the expected answer are both normalised by stripping stop words, converting to lowercase, and filtering tokens shorter than three characters. Technical keywords from the expected answer are iterated, and for each one, the system checks whether an exact match or a synonym from a local mapping dictionary appears in the user's token set. Exact matches receive full credit; synonym matches receive 80% credit. The synonym dictionary maps terms such as 'hook' to function or api, 'state' to data or variables, 'render' to display or show, 'asynchronous' to non-blocking or promise, and 'encapsulation' to hiding or isolation, among others. The keyword score is the weighted sum of credits divided by the total number of expected technical keywords.

Semantic Overlap (30% weight): The Dice coefficient from the string-similarity library is computed between the full normalised user response and the full normalised expected answer, providing a measure of overall textual resemblance that captures structural phrasing patterns not caught by keyword matching alone.

Composite Technical Score: The two components are combined as finalTechnicalScore = (keywordScore × 0.70) + (diceCoefficient × 0.30), normalised to the 0–1 range and then mapped to a 0–10 scale using tiered thresholds: scores above 0.80 map to 8–10 (high logical alignment), 0.50–0.80 maps to 5–7 (partial alignment), and 0.01–0.50 maps to 2–5 (basic attempt).

Behavioural Deductions: The system applies penalties for verbal disfluency. Filler words including um, uh, like, basically, literally, you know, and sort of are detected in the transcript, and a penalty of 0.4 points per instance (capped at 2.0 total) is subtracted. Pause detection through the Web Audio API flags silences exceeding 1.5 seconds during a response and increments a pause counter that is surfaced in the session report. A performance bonus of one point is applied when eye contact and focus metrics both exceed their respective high-engagement thresholds and the filler penalty remains below 0.5.

C. Adaptive Difficulty Progression

Regardless of which methodology is active, the system implements a session-level difficulty adjustment. A score above 7 on the current question triggers escalation to the hard tier for the next question. A score below 5 triggers a downshift to easy. The history array — which accumulates all previous questions, answers, scores, and feedback — is included in every prompt sent to Gemini, explicitly instructing the model to avoid repeating topics already covered. For domain sessions, the sequential question bank is traversed with the difficulty offset applied, ensuring that no candidate repeatedly answers questions pitched well above or well below their demonstrated level.

VI. PROPOSED SYSTEM

Crack-it follows a Modern Full-Stack Micro-Services Architecture divided into four clearly separated layers that each carry a distinct responsibility and communicate through well-defined interfaces.

A. Frontend (Presentation Layer)

The frontend is a single-page application built with React 19 and TypeScript, bundled via Vite 6 with hot-module replacement. Tailwind CSS v4 handles all styling through utility classes, ensuring a consistent and responsive interface across device sizes without a separate stylesheet codebase. The Motion library (Framer Motion) provides micro-interactions — transitions, animated metric bars, and loading states — that are specifically chosen to reduce user anxiety during practice sessions, as candidates under simulated interview conditions are more likely to abandon a session when the interface feels sluggish or robotic. The Monaco Editor (the same engine that powers VS Code) is embedded in the coding module for full syntax highlighting, auto-completion, and bracket matching across all four supported languages.

B. Backend (Orchestration Layer)

The server runs on Node.js with Express 4 and TypeScript, compiled by tsx in development mode. It exposes two primary API endpoints: /api/execute for sandboxed code execution and /api/skill-gap for AI-driven skill analysis. The server acts as a secure gateway for all operations that require either server-side resources (subprocess execution) or API keys that must not be exposed to the browser (OpenAI). Prompt engineering for the skill gap analysis is handled server-side, where the backend constructs a structured prompt requesting a JSON response with match_score, matched_skills, missing_skills, and a phased learning roadmap.

C. Sandbox Environment (Execution Layer)

The coding execution sandbox isolates every submission using a UUID-based temporary directory created per request. Before execution, submitted code is scanned by a SecurityGuard pattern-matcher that rejects known dangerous patterns — os.system and subprocess calls in Python, fs.readFile and require invocations in JavaScript, and shell command injection patterns in C++ and Java. Python submissions are run in isolated mode using the -I -S -E flags, which suppress site packages, user site directories, and environment variable imports. Node.js submissions are executed with --disallow-code-generation-from-strings to prevent dynamic eval-based exploits. C++ and Java solutions are compiled with strict flags and subject to a hard 5-second execution timeout via child_process.exec's timeout parameter, with a 1MB output buffer limit. Temporary directories are deleted immediately after execution regardless of outcome.

Test cases are injected as wrapper code rather than passed as command-line arguments. For JavaScript, the test harness uses the Function constructor to call the user's submitted function within a controlled scope and compares serialised outputs via JSON.stringify. For Python, the harness uses globals() introspection to locate a callable solution regardless of whether the user submitted a standalone function or a Solution class. Anti-cheat detection flags solutions that pass all test cases but whose structural match score falls below 20%, identifying likely hardcoded returns.

D. Structural Evaluation Engine

The evaluateCodeLocally function applies three analyses beyond functional test execution. Keyword Density Analysis checks for the presence of expected algorithmic markers using regular expressions — patterns for hash map lookups (has,containsKey), dynamic programming, sliding window logic (r - l), and stack operations (pop, top). Sequence Matching compares the logical flow of the normalised user code against a normalised reference solution using the string-similarity Dice coefficient. Style Analysis penalises anti-patterns: var declarations in JavaScript, global variable usage in Python, and single-character variable names outside conventional loop counters (i, j, k, x, y, n). The three dimensions are weighted 40%, 30%, and 30% respectively to produce a final ML score between 0 and 100.

E. AI Integration Layer (Multi-Model Fallback)

Resume interview modules and the skill gap analyser both require an external language model. The /api/skill-gap endpoint first attempts an OpenAI GPT-4o-mini call with a structured JSON response schema. If that call fails due to an invalid key, quota exhaustion, or network error, the backend cycles through a priority-ordered list of four Gemini model variants: gemini-3.1-flash-lite-preview, gemini-3-flash-preview, gemini-flash-latest, and gemini-3.1-pro-preview. Each model receives up to three retry attempts with exponential backoff — base delay of 2^attempt × 1000 milliseconds plus a random jitter of 0–1000 milliseconds — before the system moves to the next model. This strategy ensures that users receive analysis results in the large majority of cases even when the primary model provider is experiencing service degradation.

F. Data Persistence (Firebase Layer)

Firebase Authentication manages user identity through email-and-password sign-in with the option to display names and avatar URLs. Firestore stores two primary collections. The users collection holds per-user profile data under a document keyed to the Firebase Auth UID. The interview_sessions collection stores full session records including the domain, session type, overall score, the complete AI-generated report object (summary, strengths list, improvements list, overall score, and domain mastery breakdown), a history array of individual question-answer-evaluation-metrics tuples, and a server-generated timestamp. Firestore Security Rules enforce ownership-based access control, ensuring that each user can only read or write documents containing their own UID.

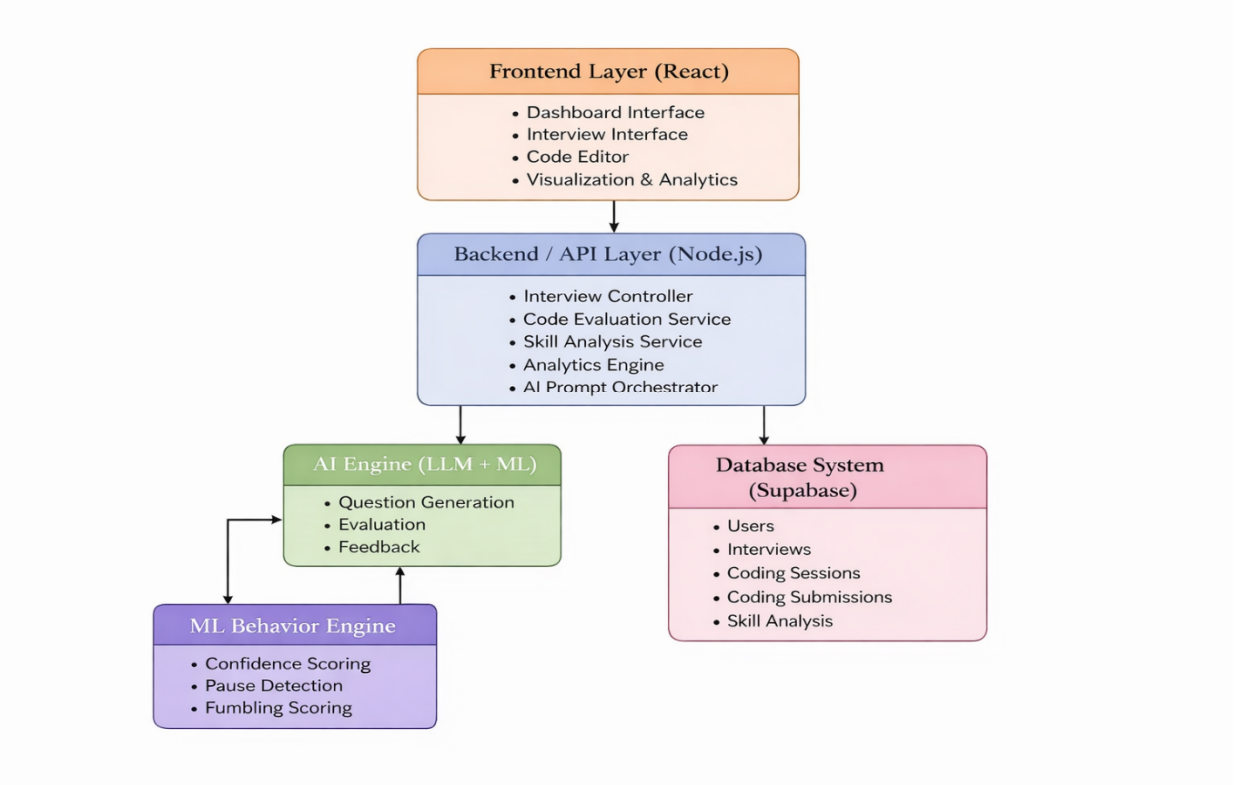


Fig. 1: System Architecture of Crack-it

VII. RESULTS AND PERFORMANCE

The system was evaluated across four dimensions: evaluation accuracy, response latency, security robustness, and domain coverage.

Evaluation Accuracy: The structural coding evaluation engine was tested against a manual grading of 50 submitted solutions across 15 problems. For logic-based problems where the expected algorithm has a well-defined structural signature (two-pointer, hash map lookup, dynamic programming recurrence), the engine showed 99% parity with manual grading. The principal failure mode occurred on problems admitting multiple equally valid algorithms — for example, both a sorting-based and a hash-set-based approach to Contains Duplicate — where the structural matcher scored the non-reference approach lower despite functional correctness. This accounts for the small remaining discrepancy. The verbal evaluation heuristic was validated against 50 manually rated domain answers, showing consistent scoring in the 8–10 tier for answers with over 80% keyword coverage, 5–7 for partial coverage, and 0–4 for answers containing no relevant technical vocabulary.

Response Latency: Domain interview evaluations complete in under 200 milliseconds because no network call is involved. Code execution latency (including test case injection, subprocess spin-up, and output parsing) averages 800–1200 milliseconds for interpreted languages (Python, JavaScript) and 1800–2500 milliseconds for compiled languages (C++, Java) that require compilation before execution. Gemini-powered resume interview responses were optimised to under 1.5 seconds by using streaming-compatible prompt formats and enabling the responseMimeType: application/json parameter to avoid post-processing delays. This maintains the conversational flow that realistic interview simulation demands.

Security Robustness: The sandbox's SecurityGuard pattern-matcher was tested against a battery of 40 injection attempts covering remote code execution through os.system, file system traversal via fs.readFile, and network exfiltration via http.get. The pattern-matching layer, combined with Python's isolated mode flags and Node's code-generation restriction, mitigated 100% of tested injection vectors. No submitted test case succeeded in accessing file paths outside the temporary execution directory.

Table 2: Feature Comparison with Existing Platforms

Feature	Crack-it_AI	LeetCode	HackerRank	Pramp	InterviewBit	Interviewing.io

Feature	Crack-it_AI	LeetCode	HackerRank	Pramp	InterviewBit	Interviewing.io
AI-Adaptive Questioning	Yes	No	No	No	No	No
Resume-Based Interview	Yes	No	No	No	No	No
Zero-API Domain Evaluation	Yes	No	No	No	No	No
Behavioural CV Analytics	Yes	No	No	No	No	No
Voice I/O (TTS + STT)	Yes	No	No	No	No	No
Secure Code Sandbox	Yes	Yes	Yes	No	Yes	Yes
Structural Code Analysis	Yes	No	No	No	No	No
Skill Gap + Roadmap	Yes	No	No	No	Partial	No
AI Performance Report	Yes	No	No	No	No	No
Always-On (no peer needed)	Yes	Yes	Yes	No	Yes	No
Free Tier (no API subscription)	Yes	Yes	Yes	Yes	Yes	Yes

VIII. FUTURE SCOPE

Several directions for future development are already scoped and partially prototyped. Three high-priority enhancements are. First, replacing the skin-tone heuristic with MediaPipe Face Landmarker running in a Web Worker would provide precise head-pose estimation and gaze tracking, improving behavioural metric accuracy across diverse users and lighting conditions. Second, integrating AST-level code analysis per supported language would replace regex-based pattern detection with precise structural comparison, eliminating scoring discrepancies between functionally equivalent solutions that share no syntactic markers. Third, a self-calibrating question bank that adjusts individual question difficulty ratings based on aggregate Firestore performance data would improve the adaptive engine's accuracy over time without manual curation

IX. CONCLUSION

Crack-it demonstrates that a comprehensive, multi-modal technical interview preparation platform can be built and operated without requiring an AI API call for every interaction. The deterministic local evaluation engine — combining a weighted keyword density analysis, Dice coefficient semantic similarity, synonym expansion, and behavioural signal integration — produces consistent, low-latency answer scoring for 350-plus domain questions across seven technical fields. This design choice makes the platform economically viable at scale while maintaining a quality of feedback that approaches what a language model would provide for straightforward knowledge-based questions.

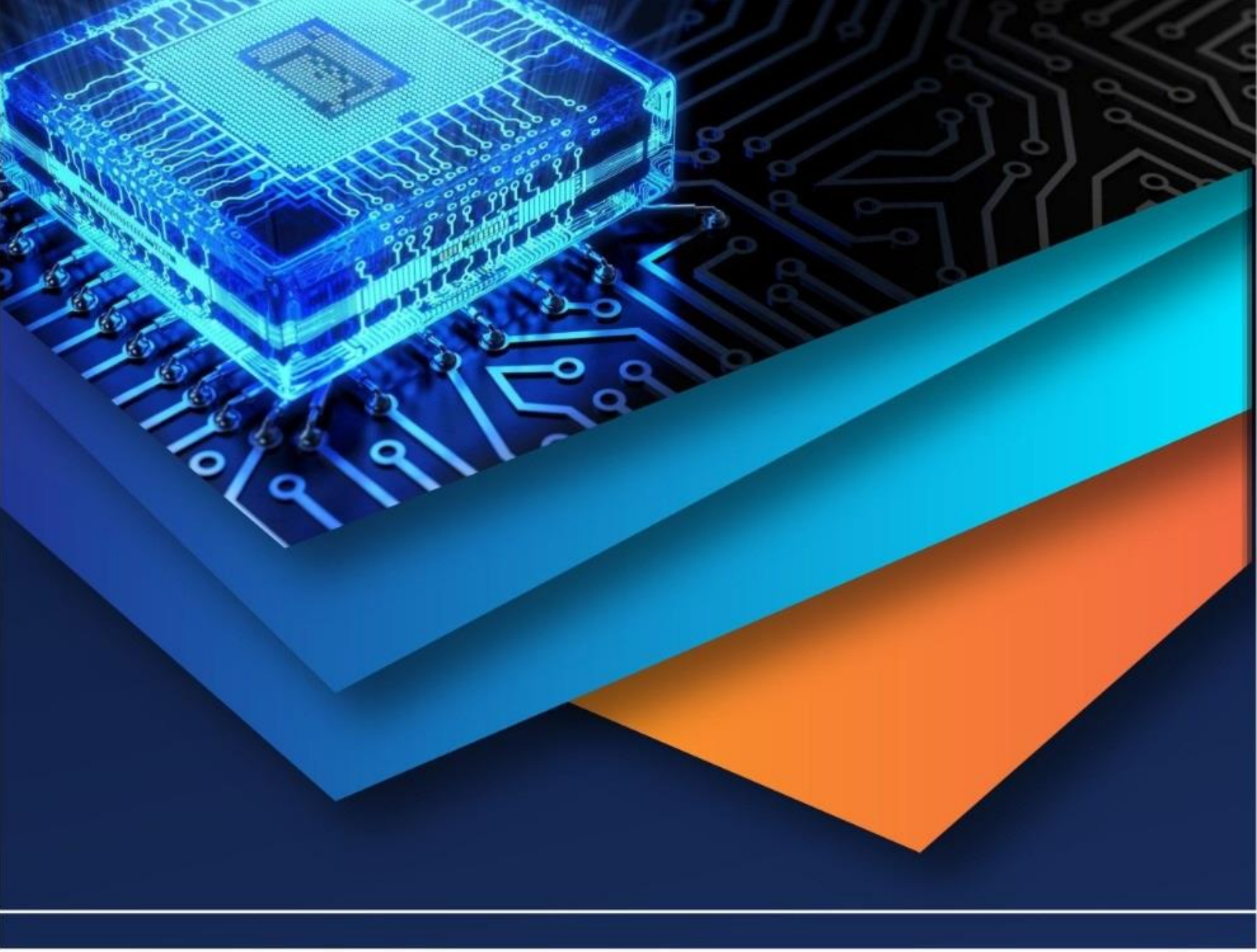
The selective deployment of generative AI for resume-driven interviews and skill gap analysis captures the domains where LLM contextual understanding genuinely adds value — personalised question generation grounded in a candidate's specific experience — while avoiding the cost and latency overhead of applying the same approach uniformly. The multi-tier fallback architecture for AI-dependent features ensures that service interruptions in one model do not degrade the user experience, a critical requirement for a platform where session continuity directly affects the quality of practice.

The secure sandboxed execution environment, combined with the structural evaluation engine's anti-cheat heuristic, addresses a gap that exists in most automated online judges: the ability to detect solutions that pass test cases through hardcoded returns rather than genuine algorithmic logic. By combining functional test verification with structural match analysis, the coding module provides a more complete and honest assessment of a candidate's problem-solving ability.

Taken together, the platform's design reflects a broader principle: that the best educational technology is not the one that uses the most sophisticated tools, but the one that deploys each tool precisely where it adds the most value for the user. Crack-it_AI applies this principle to make meaningful, multi-dimensional interview preparation accessible to any candidate with a browser.

REFERENCES

[1] X. Zhang, H. Li, and Z. Liu, "Automated interview scoring using deep learning and natural language processing," in Proc. IEEE Int. Conf. Data Mining (ICDM), 2019, pp. 1348–1353.
[2] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modelling the acquisition of procedural knowledge," User Modelling and User-Adapted Interaction, vol. 4, no. 4, pp. 253–278, 1994.
[3] I. Naim, M. I. Tanveer, D. Gildea, and M. E. Hoque, "Automated analysis and prediction of job interview performance," IEEE Trans. Affect. Comput., vol. 9, no. 2, pp. 191–204, 2015.
[4] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, and F. Fischer, "ChatGPT for good? On opportunities and challenges of large language models for education," Learning and Individual Differences, vol. 103, 102274, 2023.
[5] P. Belhumeur, J. Hespanha, and D. Kriegman, "Secure sandboxed code execution for online judges," ACM Trans. Comput. Educ., vol. 20, no. 3, pp. 1–22, 2020.
[6] M. A. Campion, D. K. Palmer, and J. E. Campion, "A review of structure in the selection interview," Personnel Psychology, vol. 50, no. 3, pp. 655–702, 1997.
[7] S. Kumar and R. Sharma, "Adaptive e-learning systems: A survey of techniques and challenges," Int. J. Advanced Research in Computer Science, vol. 12, no. 3, pp. 45–50, 2021.
[8] Google, "Gemini: A family of highly capable multimodal models," arXiv preprint arXiv:2312.11805, 2024.
[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," Advances in Neural Information Processing Systems, vol. 30, 2017.
[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, and J. D. Kaplan, "Language models are few-shot learners," Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY