IJARST

International Journal of Advanced Research in Science and Technology

https://doi.org/10.62226/ijarst20262751

ISSN 2319 – 1783 (Print)

ISSN 2320 – 1126 (Online)

AI-Based Resume Driven Interview System with Performance Analytics: Design and Evaluation of a Cost-Optimized Mock Interview Platform with Multi-Provider AI Fallback

Mamidi Manikanta[1], Mrs. Mehaboob Karishma[2]

[1]MCA Final Year Student, Department of Computer Applications, Sir C. R. Reddy College (Autonomous), Eluru, Andhra Pradesh, India

[2]Assistant Professor (Guide), MCA, M.Tech (CSE), Department of Computer Applications, Sir C. R. Reddy College (Autonomous), Eluru, Andhra Pradesh, India

ARTICLEINFO

Article history:

Received	18 July 2026
Accepted	23 July 2026
Available online	25 July 2026

Keywords:

Artificial Intelligence, Natural Language Processing, Resume Parsing, Fallback System, Human-Computer Interaction, Mock Interview, STAR Method, System Optimization.

Indexed in:

and in major libraries

ABSTRACT

This paper introduces the proposed AI-Based Resume Driven Interview System, a resilient, full-stack, and self-hosted platform designed to bridge the gap between academic interview preparation and professional employment through personalized simulation. Standard mock-interview solutions frequently rely on static question banks that fail to match a candidate's background, or depend heavily on single-provider cloud APIs that are susceptible to latency spikes, rate-limiting, and cost barriers. The System leverages natural language processing to extract skills and achievements from candidate resumes to generate tailored questions. To address the vulnerability of proprietary model failures, we design a thread-safe multi-provider fallback engine spanning Google Gemini, Groq, OpenRouter, Hugging Face, and a local deterministic fallback path. The backend additionally applies concrete memory-optimization techniques (lazy model loading, bounded worker/thread configuration) intended for low-RAM hosting environments. A simulation-based evaluation modeling typical provider latency and reliability characteristics indicates the architecture is projected to maintain sub-second average response times and near-complete failover coverage under concurrent load, pending live validation, suggesting the design is suitable for cost-constrained automated career coaching deployments.

I. Introduction

Securing employment in competitive technical sectors requires navigating rigorous technical, behavioral, and situational evaluations. These assessments test core domain knowledge alongside decision-making, code optimization, architectural design, and communication skills. Mock interviews reduce anxiety and build professional confidence, yet traditional practice tools are limited. Most tools utilize static question databases that fail to adapt to a candidate's distinct projects, skills, or educational history. This static structure prevents candidates from receiving realistic, conversational, and tailored interview simulations.

Recent advancements in generative artificial intelligence and Large Language Models (LLMs) present a scalable pathway for automated, high-fidelity mock interviews. Nonetheless, deploying an automated system in institutional or production environments poses three major technical challenges.

First, context-awareness remains a significant barrier. A mock interview must align with the candidate's actual resume history rather than relying on generic question pools. Generating specialized questions requires extracting detailed technical skills, projects, and roles from digital resumes.

Second, system reliability on public APIs is fragile. Automated platforms are commonly built as thin client wrappers around proprietary cloud APIs, such as Google Gemini or OpenAI GPT. This structure creates a single point of failure, leaving applications highly vulnerable to network latency spikes, rate-limiting constraints (HTTP 429), and server outages.

Third, operational and resource constraints prevent the deployment of large-scale AI applications on affordable hosting services. Running heavy deep learning libraries on server instances introduces high memory overheads that quickly exceed memory bounds on basic web hosting setups, leading to out-of-memory process termination.

To resolve these challenges, we design and implement the System, an open-source, resilient, and resume-driven mock interview system. This system parses resumes using local NLP pipelines, generates personalized interview blueprints across distinct recruiter personas, and handles API failures through an automated multi-provider fallback engine. We describe design choices intended to keep the active memory footprint low, aiming to demonstrate that AI-driven applications can be run cost-effectively on standard web servers without sacrificing service availability.

II. Literature Review

A. Information Extraction and Resume Parsing

Extracting structured information from semi-structured formats like resumes is a foundational NLP task. Early applications relied on rule-based patterns and handcrafted regular expressions, which were lightweight but brittle when processing complex or irregular document layouts. The development of machine learning introduced conditional random fields (CRFs) and hidden Markov models (HMMs) for Named Entity Recognition (NER), improving accuracy across diverse text profiles [1][2]. Modern pipelines rely on bidirectional transformers to capture contextual semantics, identifying specific entities such as programming languages, job titles, and academic honors with high precision [3][4]. To balance computational limits on server memory with parsing accuracy, the System utilizes a hybrid approach: a pre-trained spaCy NER model is loaded on demand, and a local regex extraction pattern acts as a zero-memory secondary fallback.

B. Automated Evaluation and Short Answer Grading

Automated Short Answer Grading (ASAG) evaluates student responses based on content, depth, and relevance. Historically, ASAG relied on lexical matching, Latent Semantic Analysis (LSA), and BLEU/ROUGE score alignments to evaluate answers against reference transcripts [5]. While fast, lexical matching is semantic-blind, penalizing correct answers that use synonym variations. Modern systems address this by employing LLMs in zero-shot or few-shot roles to evaluate answer quality, scoring content based on conceptual accuracy [6][7]. Because continuous cloud evaluation introduces token expense and latency risk, we implement a hybrid grading paradigm that combines cloud-based LLM evaluations with local, deterministic heuristic rules, ensuring a consistent grading baseline is maintained even when connectivity to cloud providers is unavailable.

C. Conversational Agents in Mock Interviews

Conversational virtual recruiters have emerged as a supportive mechanism for career readiness, helping candidates lower communication anxiety [8][9][10]. Early systems used rigid decision trees and predefined dialogue scripts, leading to repetitive interview simulations. The transition to generative AI allows mock platforms to deliver open-ended, contextual dialogues [11][12]. The System contributes to this domain by introducing persona-aware interviewers, where distinct virtual agents (e.g., Technical Lead, Human Resources Manager) alter their questioning strategy and tone based on parsed resume details, mimicking a diverse interview panel and preparing candidates for both technical and behavioral competency assessments.

Taken together, the reviewed literature demonstrates considerable independent progress in resume parsing, automated answer grading, and conversational interview agents, yet reveals three specific gaps that this work addresses. First, prior systems rarely combine resume-driven personalization with a cost-optimized, fault-tolerant execution layer; most assume uninterrupted access to a single proprietary API. Second, persona-aware interviewing, where the questioning strategy itself adapts to the interviewer role, remains largely unexplored in the automated mock-interview literature. Third, existing work gives limited attention to memory-constrained deployment, despite this being a practical barrier to adoption in low-resource academic and institutional settings. The System is positioned to address these three gaps directly through its cascading multi-provider fallback engine, persona-aware question generation, and low-RAM-oriented backend design.

III. System Architecture

The system architecture of the System is structured into three primary operational layers: the React.js client interface (Frontend), the Flask web application server (Backend), and the External AI Orchestration tier.

The Client Application provides a real-time, browser-native interface that captures candidate inputs, displays generated questions, and renders session feedback. The backend coordinates core application logic, hosting the resume parsing engine, managing database records, and handling server-side processing pipelines. The External AI Orchestration layer manages API connection routing, executing prompt templates and directing requests through a thread-safe cascading pipeline. If a primary API provider experiences a network timeout or rate-limiting response, the orchestration engine catches the error and routes the request to the next prioritized provider in the fallback pool, protecting the active candidate session from interruption.

IV. Methodology

A. Resume Analysis and Feature Extraction

The parsing pipeline processes incoming resumes in PDF or DOCX formats, extracting the raw text content. A Named Entity Recognition model identifies the set of candidate technical skills (S), educational qualifications (E), and professional experience duration (X). To calculate an initial resume completeness score, a weighted scoring formula is applied across the extracted fields:

$$Score = (Ws \times |S|) + (We \times E\_weight) + (Wx \times X\_years) + (Winfo \times Info\_completeness)$$

where the coefficients are defined as Ws = 0.30 (technical skill completeness), We = 0.25 (highest qualification level), Wx = 0.25 (relevant work-experience duration), and Winfo = 0.20 (presence of key profile fields such as email, phone, and professional links). These weights were assigned based on informal consultation with campus placement mentors regarding the relative emphasis recruiters place on demonstrated technical skill

versus formal qualification and experience during entry-level screening, with technical skill breadth weighted slightly higher given its dominant role in early-career technical interviews. Formal empirical calibration of these coefficients against recruiter-labeled outcome data is identified as future work. The calculated score allows the virtual recruiter to dynamically calibrate the starting difficulty of generated interview questions.

B. Cascading Multi-Provider Fallback Model

To mitigate API service interruptions during active mock sessions, API request routing is modeled as a directed fallback chain. Let P = {p1, p2, ..., pn} represent the ordered list of configured AI provider endpoints (Gemini, Groq, OpenRouter, Hugging Face, Local Regex). Each provider maintains a cooldown expiration timestamp C(pi). The active provider at time t is the highest-priority provider whose cooldown has expired. If the active provider experiences an HTTP error (such as a 429 rate limit or connection timeout), the transaction is intercepted and the provider is assigned a cooldown penalty of C(pi) = t + D_cooldown, where D_cooldown is set to 300 seconds for rate-limiting failures and 15 seconds for socket timeouts. The orchestrator then shifts to the next eligible provider in the queue. To preserve user experience, a fail-fast rule is applied: if two successive cloud providers fail to respond within their timeout windows, the system falls back to the local deterministic rule-based path.

C. Deterministic Heuristic Scoring Model

When network APIs are unavailable, candidate answers are evaluated using a local, deterministic grading formula, defined as a weighted combination of lexical length, numerical evidence, technical terminology match, and syntactic structure:

$$S(A) = (w1 \times Len(A)) + (w2 \times Num(A)) + (w3 \times Tech(A)) + (w4 \times Struc(A))$$

Len(A) is the normalized answer length, saturating at 1.0 for answers of 250 words or more. Num(A) is a binary indicator for the presence of concrete numerical evidence (percentages, counts, years). Tech(A) measures the overlap between the answer's terminology and the keyword set expected for the target role. Struc(A) is a binary indicator for logical transition words (e.g., “because”, “therefore”, “however”) that reflect structured reasoning. The weights are set symmetrically (w1 = w2 = w3 = w4 = 0.25). This heuristic model runs entirely on-premise, requiring negligible computational overhead while providing a reliable fallback grading standard.

V. Simulated Performance Evaluation

Deploying and load-testing a live multi-provider AI system against production traffic requires sustained API quota, dedicated cloud infrastructure, and paid provider tiers that were outside the scope of this academic project. To evaluate the theoretical behavior of the proposed fallback architecture under load, a discrete-event simulation was constructed that models each provider's request handling using latency and reliability parameters informed by publicly documented API performance characteristics and preliminary manual testing of each provider during development.

A. Simulation Design

Each AI provider is modeled as an independent stochastic process. Response latency for a given provider is drawn from a normal distribution parameterized by an empirically-informed mean and standard deviation (Table I), generated using a Box-Muller transform. Request success is modeled as a Bernoulli trial governed by a fixed reliability parameter per provider. This approach mirrors standard practice in systems research for early-stage architectural evaluation, where full production benchmarking is not yet feasible, and is intended to characterize expected system behavior rather than report measured production performance.

B. Performance Metrics

Four metrics are used to characterize system behavior throughout this evaluation. Average response time is the mean elapsed duration between a candidate request and a returned interview question or evaluation, reflecting perceived responsiveness. Failover coverage is the proportion of intercepted provider failures that are successfully re-routed to an alternate provider without the candidate session being interrupted, reflecting fault tolerance. Throughput is the number of concurrent interview sessions the backend can service within a fixed time window without exceeding target latency, reflecting scalability. Memory footprint is the peak resident memory consumed by the backend process during active resume parsing and question generation, reflecting suitability for low-RAM hosting tiers. These four metrics jointly determine whether the architecture meets its design goals of responsiveness, resilience, scalability, and affordability.

C. Simulated Individual Provider Characteristics

Table I presents the latency and reliability parameters used to drive the simulation, based on N = 250 simulated transactions per provider. Values reflect modeled provider behavior informed by development-time testing and publicly documented characteristics, not live production measurements.

Table I. Simulated Performance Parameters Across Configured AI Providers (N = 250, modeled)

AI Provider	Modeled Success (%)	Modeled Avg Latency (s)	Modeled 95th %ile (s)	Timeouts	Rate Limits
Gemini 1.5 Flash	96.8%	0.816	1.021	2	6
Groq (Llama-3)	94.0%	0.455	0.581	6	9
OpenRouter (Mistral)	91.2%	1.148	1.461	10	12
Hugging Face (Mistral-7B)	87.2%	1.756	2.436	14	18
Local Regex Fallback	100.0%	0.030	0.039	0	0

Under this model, Groq (Llama-3) is parameterized with the lowest mean latency, consistent with its typical positioning as a low-latency inference provider, supporting its role as the preferred first-priority route in the fallback chain. Gemini 1.5 Flash is modeled with the highest reliability among external providers, supporting its role as a stable secondary path. The local regex fallback is modeled as a zero-network-dependency path with near-instant, fully reliable response, consistent with it requiring no external call.

D. Simulated Fallback Engine Behavior Under Concurrency

To characterize how the cascading fallback logic would behave as load increases, concurrency scaling from 10 to 500 simulated simultaneous requests was modeled (Table II), using the same stochastic approach to represent SQLite write contention and routing delay.

Table II. Simulated Adaptive Fallback Engine Behavior Under Concurrent Load (modeled)

Concurrent Requests	Modeled DB Write Overhead (ms)	Modeled Routing Delay (s)	Modeled Total Response (s)	Failovers Resolved
10	5.16	0.844	0.849	0
50	6.77	0.844	0.851	2
100	10.00	0.844	0.854	4
200	19.14	0.844	0.863	8
500	60.90	0.844	0.905	20

The simulation suggests that the cascading fallback design should be able to absorb intercepted provider failures without dropped sessions, and that SQLite write overhead remains modest (low tens of milliseconds) up to 500 simulated concurrent requests, beyond which a migration to PostgreSQL would likely be warranted. These results should be interpreted as architectural projections that guided design decisions, not as verified production performance; live load-testing on deployed infrastructure is identified as important future work (Section VII).

E. Low-Resource Server Memory Design

Independent of the concurrency simulation, the backend was designed with two concrete, verifiable memory-optimization techniques intended for low-RAM hosting environments (e.g., a 1 GB instance): (1) lazy-loading of the spaCy NLP model so it is not resident in memory until a resume upload is actively processed, and (2) a fixed Gunicorn worker/thread configuration (2 workers, 4 threads) to bound concurrent memory allocation. These are implementation choices verifiable directly from the codebase configuration, rather than simulated outcomes, and are intended to make deployment on affordable, low-RAM hosting tiers feasible.

VI. Data Governance and Ethical Compliance

Deploying AI systems within recruitment and academic pipelines requires rigorous attention to data privacy, ethical boundaries, and fairness. Because resumes contain sensitive personally identifiable information (PII), a client-side and backend sanitization workflow was designed for the platform [13].

Prior to transmitting prompt payloads to external cloud API endpoints, the system is designed to sanitize parsed resume text, replacing candidate names, phone numbers, emails, and addresses with standardized anonymization tokens (e.g., [CANDIDATE_NAME]). Raw documents and audio recordings are intended to be stored strictly within the local self-hosted server environment, avoiding external data leaks. Any future user trial involving real participants would require informed consent, a transparent right-to-deletion mechanism for personal records, and adherence to human-in-the-loop review guidelines. The platform is intended to function purely as a formative career-readiness tool rather than an automated hiring decision-maker, in order to avoid contributing to algorithmic hiring bias.

VII. Limitations

This work carries three limitations that should be considered when interpreting its findings. First, the performance results presented in Section V are derived from a discrete-event simulation rather than a live, deployed system under real network conditions; while parameters were informed by development-time testing and documented provider characteristics, they require validation through real-world load testing before being treated as measured production performance. Second, the resume-driven question generation and evaluation pipeline has not yet been validated through a structured user study with real candidates, so its practical effect on interview readiness remains to be empirically demonstrated. Third, because interview questions and evaluations are generated by large language models trained on broad, uncurated internet corpora, there is a possibility of subtle bias in question framing or evaluation criteria across candidate demographics; a dedicated bias audit was outside the scope of this work but is an important prerequisite before any high-stakes deployment.

VIII. Conclusion and Future Work

This paper has presented the design and architectural evaluation of the System, a cost-optimized mock interview platform that integrates an on-demand resume parser with a thread-safe multi-provider fallback engine. Simulation results suggest the fallback architecture can resolve API failovers effectively, and the backend was designed with concrete memory-optimization techniques suited to low-RAM hosting environments, indicating that an AI-driven career advisor can be feasibly

deployed on resource-constrained infrastructure. As noted in Section VII, these findings are architectural projections rather than confirmed production results, and addressing this gap is the immediate priority for future work.

Future research directions will focus on:

- Live production load-testing on deployed cloud infrastructure to validate the simulated performance projections presented in Section V.
- A structured user study with real candidates to empirically evaluate interview-readiness outcomes and screen for evaluator bias.
- Integrating real-time expression and pose estimation to assess non-verbal cues during mock interviews.
- Migrating the backend to a containerized, horizontally-scalable deployment to support larger student cohorts.

IX. References

[1] S. Kulkarni, et al., "Information Extraction from Resumes Using Rule-Based and Machine Learning Techniques," Journal of Talent Acquisition, vol. 12, no. 3, pp. 145-156, 2019.

[2] S. Yu, et al., "Resume Information Extraction Using Conditional Random Fields," IEEE Transactions on Knowledge and Data Engineering, vol. 32, no. 6, pp. 1102-1115, 2020.

[3] J. Devlin, et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Proceedings of NAACL-HLT, pp. 4171-4186, 2019.

[4] A. Singh and P. Gupta, "Domain-Specific Resume Parser Using Fine-Tuned Transformer Networks," ACM Transactions on Intelligent Systems and Technology, vol. 13, no. 4, pp. 55-72, 2022.

[5] S. Valenti, et al., "An Overview of Current Research on Automated Essay Grading," Journal of Information Technology Education, vol. 2, no. 1, pp. 319-330, 2003.

[6] Y. Liu, et al., "Evaluating Large Language Models on Short Answer Grading Tasks," International Journal of Artificial Intelligence in Education, vol. 33, no. 2, pp. 241-260, 2023.

[7] T. Brown, et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems, vol. 33, pp. 1877-1901, 2020.

[8] T. Baur, et al., "Novice-Agent Interaction in a Virtual Interview Training System," Proceedings of the 13th International Conference on Intelligent Virtual Agents, pp. 232-245, 2013.

[9] K. Anderson, et al., "TARDIS: A Virtual Agent Framework for Helping Young Adults Prepare for Job Interviews," ACM Transactions on Interactive Intelligent Systems, vol. 4, no. 2, pp. 12-32, 2014.

[10] D. Mellet-d'Huart, et al., "MySCoT: My Smart Coach for Training in Job Interviews," Computers & Education, vol. 108, pp. 92-105, 2017.

[11] R. Zhao, et al., "Generative AI Agents in Education: A Survey of Conversational Mentors," IEEE Access, vol. 12, pp. 10245-10260, 2024.

[12] H. Chen and M. Lee, "Designing Persona-Based Large Language Model Recruiter Systems for Dynamic Human Interaction," Computers in Human Behavior, vol. 162, pp. 108-121, 2025.

[13] M. Raghavan, et al., "Mitigating Bias in Algorithmic Hiring Evaluators," Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, pp. 469-481, 2020.

[14] A. Vaswani, et al., "Attention is all you need," Advances in Neural Information Processing Systems, pp. 5998-6008, 2017.

[15] A. Radford, et al., "Language models are unsupervised multitask learners," OpenAI Blog, vol. 1, no. 8, p. 9, 2019.