Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

PrepWise AI: A Smart Preparation Assistant for Study and Career

Jagadeesh Pujari
Information Science and Engineering
SDM College of Engineering and Technology
Dharwad, India

Varsha S Jadhav
Information Science and Engineering
SDM College of Engineering and Technology
Dharwad, India

Amogh A Koujalagi
Information Science and Engineering
SDM College of Engineering and Technology
Dharwad, India

Nikhil S Joshi
Information Science and Engineering
SDM College of Engineering and Technology
Dharwad, India

Shrilakshmi A Janadri
Information Science and Engineering
SDM College of Engineering and Technology
Dharwad, India

Akshat V Kalal
Information Science and Engineering
SDM College of Engineering and Technology
Dharwad, India

***Abstract*—Modern undergraduate engineering students face a dual challenge: mastering semester syllabi within compressed academic timelines while simultaneously cultivating the technical and communicative competencies that industry recruitment processes demand. Existing digital tools tend to address only one of these dimensions in isolation, leaving students without a cohesive and adaptive support ecosystem. This paper introduces *PrepWise AI*, a full-stack, privacy-preserving intelligent platform that consolidates academic study planning and multimodal interview preparation into a single unified environment.**

The system is organized around four interdependent modules. The *Smart Study Planner* interprets free-form natural language exam prompts and constructs structured, day-wise preparation schedules through constrained prompt engineering over a locally hosted large language model (LLM). The *Internal Assessment (IA) Planner* bridges teacher-defined institutional syllabi with student-facing AI-generated resources, dynamically calibrating content depth and emphasis based on examination proximity. The *Resources Module* delivers curated theory notes enriched with diagrams alongside an embedded Data Structures and Algorithms (DSA) coding environment powered by the Monaco Editor, with real-time test-case evaluation. The *AI Interview Assistant* conducts comprehensive multimodal candidate evaluation through synchronized video analysis—comprising MediaPipe Face Mesh landmark detection, pixel-level frame processing, and YOLO-based object detection—alongside audio preprocessing using FFmpeg for extraction, resampling, and normalization of the audio signal, followed by pitch estimation and speech rate computation, culminating in semantic answer evaluation via a locally deployed LLM.

The backend is implemented using Flask and FastAPI microservices. All LLM inference executes on-premise through the Ollama platform using Qwen 2.5 (7B) and Mistral models. Firebase Firestore supports institutional data persistence. Experimental evaluation with thirty undergraduate participants yielded a schedule relevance score of 86.4%, a user satisfaction rating of 4.2 out of 5.0, and a plan completion rate of 72.4%. Interview assessment quality was validated by a Pearson correlation of $r = 0.81$ with expert human raters, accompanied by a semantic evaluation accuracy of 84.7%, speech clarity consistency of 87.2%, and speech-to-text accuracy of 91.3%, at a mean LLM response latency of 4.2 seconds. These results collectively confirm that PrepWise AI constitutes a practical, scalable, and pedagogically sound contribution to AI-assisted academic and career preparation.

***Index Terms*—Large Language Models, Adaptive Study Planning, AI Interview Assessment, Multimodal Behavioral Analysis, Speech Processing, FFMPEG, Face Mesh, YOLO Object Detection, Ollama, Monaco Editor, Firebase, React, Flask, FastAPI**

I. Introduction

The educational landscape for undergraduate engineering students has grown progressively more demanding over the past decade. Syllabi have expanded in scope and interdisciplinary depth, internal assessment schedules have become increasingly compressed, and industry recruitment benchmarks have shifted from rote knowledge recall toward demonstrating applied problem-solving under timed, observed conditions. A student preparing for a Database Management Systems examination five days away faces a fundamentally different challenge compared with one practising for a technical interview scheduled the following morning, yet both share the same underlying need: timely, relevant, and personalized guidance that adapts to their specific constraints.

Conventional digital study tools fall broadly into two categories. Static content platforms—online video courses, PDF note repositories, and textbook companions—deliver fixed material at a uniform depth irrespective of the learner's timeline or prior knowledge. Adaptive learning systems, while more sophisticated, have historically required extensive user

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

profiling, labelled interaction datasets, and cloud infrastructure that raises significant data privacy concerns for educational institutions. Neither category addresses the practical reality of a student who wishes to type a natural sentence such as *"I have an Operating Systems exam in six days"* and receive an immediately actionable, content-rich study plan that also teaches the material chapter by chapter.

The recent emergence of open-weight large language models deployable on consumer-grade hardware—including LLaMA, Mistral, and Qwen—served through lightweight inference platforms such as Ollama, has fundamentally altered the feasibility of on-premise intelligent tutoring systems. These models enable institutions and individual developers to build sophisticated natural language interfaces without transmitting sensitive academic data to third-party cloud providers. Concurrently, advances in browser-native computer vision through WebAssembly-compiled libraries such as MediaPipe, real-time audio processing in Python, and browser-embedded code editors (Monaco) have made it technically viable to deliver IDE-quality coding environments and comprehensive behavioral interview assessments entirely within a standard web browser session.

PrepWise AI synthesizes these converging capabilities into a single cohesive platform designed specifically for undergraduate engineering students. The system architecture spans four functional modules that together address the full preparation lifecycle: structured exam planning, institutional schedule integration with AI-generated content, subject-matter learning with hands-on coding practice, and realistic mock interview simulation with detailed behavioral and semantic performance feedback.

Three design principles govern the system. *Adaptability*: content structure and depth respond dynamically to the user's specific timeline and urgency level. *Privacy*: all AI inference executes locally, with no student data transmitted to external services. *Comprehensiveness*: the platform functions as a one-stop preparation environment rather than a single-purpose tool, reducing the cognitive overhead of switching between disparate applications.

The primary contributions of this work are as follows:

- A constrained prompt engineering framework that converts free-form natural language exam prompts into deterministically structured, day-wise study plans using locally hosted LLMs, with guaranteed reservation of the final preparation day for revision and mock testing.
- An urgency-tiered IA Planner that integrates Firebase Firestore for persistent institutional data with a Flask-proxied Ollama LLM, dynamically reconfiguring prompt depth and content focus across three examination proximity tiers.
- A multimodal AI Interview Assistant that fuses pixel-level video frame analysis, MediaPipe Face Mesh landmark tracking, YOLO-based prohibited-object detection, FFMPEG-based audio extraction and resampling, Whisper speech-to-text transcription, and LLM semantic scoring into a single weighted Interview Performance Score (IPS).
- An embedded DSA practice environment using the Monaco Editor that supports multi-language code execution and real-time test-case evaluation without requiring external platform accounts.
- Quantitative validation through user studies with undergraduate participants, demonstrating strong agreement between automated and expert human assessments.

The remainder of this paper is organized as follows. Section II reviews related literature. Section III describes the overall system architecture. Section IV details the methodology of each module. Section V reports and analyzes experimental results. Sections VI through VIII discuss advantages, limitations, and future directions. Section IX concludes the paper.

II. Literature Review

A. Adaptive Learning and Personalized Study Planning

The theoretical foundation for adaptive instructional sequencing can be traced to the intelligent tutoring systems developed in the late 1970s and 1980s. Corbett and Anderson [4] formalized Bayesian knowledge tracing, a probabilistic model that estimates a student's mastery of discrete skills over repeated practice events and uses these estimates to determine when a learner should advance to new material. Subsequent extensions incorporated prerequisite knowledge graphs, collaborative filtering over student cohorts, and reinforcement learning for policy optimization, producing systems capable of fine-grained, personalized content sequencing.

The rise of transformer-based large language models introduced natural language as a primary interface for study planning [1]. Rather than requiring students to complete structured profiling questionnaires, LLM-based systems can infer intent, subject scope, and timeline constraints from conversational input phrased in everyday language. Research has demonstrated that instruction-tuned models produce coherent multi-day learning plans when provided with appropriately constrained prompts; however, the same literature documents a well-known tendency toward hallucination—the generation of plausible-sounding but factually incorrect or structurally inconsistent outputs—particularly when models are not given explicit format constraints [6]. PrepWise AI addresses these shortcomings through hard structural constraints embedded in the prompt and backend format validation before delivery to the frontend.

B. Automatic Speech Recognition and Spoken Language Assessment

Automatic speech recognition has progressed substantially, from early Hidden Markov Model systems coupled with Gaussian mixture acoustic models to contemporary end-to-end deep learning architectures [14]. Whisper [7], trained through weakly supervised learning on a large multilingual corpus, achieves near-human word error rates on standard English audio and demonstrates acceptable robustness under moderate background noise. Vosk provides a lightweight,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

offline-capable alternative well-suited to environments where real-time transcription is required without cloud connectivity.

Beyond transcription, spoken language assessment research has explored rich acoustic feature sets for evaluating fluency, confidence, and delivery quality [3]. FFmpeg is used for audio preprocessing, where the audio stream is extracted from the recorded video, converted into a consistent format, and resampled to a standard frequency for further analysis. It also performs normalization to ensure uniform audio levels, improving the reliability of downstream processing. The cleaned and standardized audio is then used for pitch estimation and speech rate computation, which serve as indicators of vocal clarity, confidence, and delivery pacing. PrepWise AI extracts all three feature categories and combines them into a composite speech clarity score for integration into the IPS.

C. Computer Vision for Behavioral Interview Analysis

Video-based analysis of candidate behavior during interviews has been an active research area since the widespread adoption of video telephony and video-based hiring platforms. Early systems relied on manually engineered features—nod frequency, gaze aversion events, hand gesture amplitude—extracted using optical flow and Haar-cascade face detection [2]. These systems established statistically significant correlations between non-verbal behavioral signals and expert-rated interview performance.

More recently, deep learning face analysis frameworks have provided substantially improved facial landmark localization. MediaPipe Face Mesh [8] extracts 468 three-dimensional landmark coordinates from a monocular webcam feed in real time, enabling the computation of head pose (pitch, yaw, and roll) with sufficient accuracy for gaze direction estimation and eye-contact scoring. Crucially, this framework operates entirely in-browser via WebAssembly, requiring no server-side video transmission. YOLO architectures [9], pioneered by Redmon et al. and refined through several major versions, provide real-time single-pass object detection suitable for streaming video. Prior work has applied YOLO to online examination proctoring to detect prohibited materials and secondary persons within a candidate's field of view—a technique adopted by PrepWise AI to flag distracting or prohibited objects and incorporate a corresponding penalty into the attention score.

D. Semantic Answer Evaluation Using Large Language Models

Automated grading of open-ended short answers presents challenges that purely lexical similarity metrics such as BLEU and ROUGE cannot address, because semantically equivalent answers may share little surface-level vocabulary. Models fine-tuned on natural language inference tasks can assess semantic entailment between reference and candidate answers, while instruction-tuned generative models prompted with a rubric and the original question can produce structured grading judgments with justifications [10]. Open-weight models such as LLaMA [11] and Mistral extend this capability to on-premise deployments, which is critically important in educational contexts where student responses constitute sensitive personal data. PrepWise AI uses Ollama to serve these models locally, passing the transcribed candidate response alongside the original question in a zero-shot evaluation prompt that requests a structured score and justification.

E. Integrated Educational Platforms

Commercial platforms such as Coursera, Khan Academy, and LeetCode each specialize in one preparation dimension—video lectures, adaptive exercises, or coding challenges—but none provide the full spectrum from personalized exam scheduling through live interview assessment. Firebase has been adopted as a backend-as-a-service in several educational mobile applications owing to its real-time synchronization and flexible document model [12]. Browser-based code editors, particularly Monaco [13], have been integrated into platforms such as VS Code for the Web and several academic online judges, confirming the viability of in-browser IDE-quality environments. PrepWise AI draws on these established technologies while combining them in a novel architectural arrangement that specifically serves the integrated preparation needs of engineering undergraduates.

III. System Architecture

PrepWise AI is structured as a modular, three-tier web application in which each tier has clearly defined responsibilities and communicates with adjacent tiers through well-specified interfaces. The three tiers are: the presentation layer (React frontend), the application logic layer (Python microservices), and the data and inference layer (Firebase Firestore and Ollama-served LLMs). This separation of concerns ensures that individual modules can be updated or replaced independently without disrupting the rest of the system.

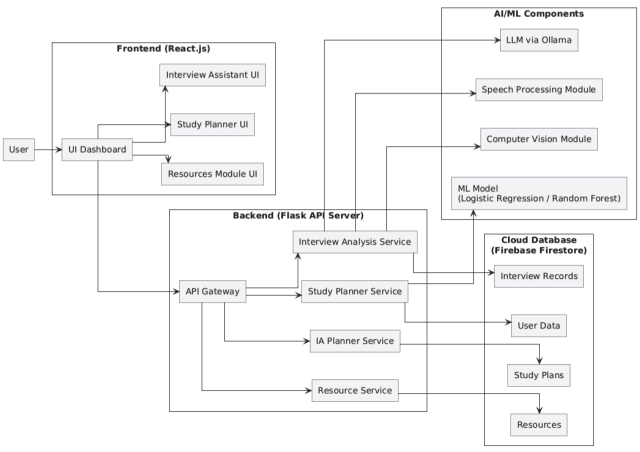

Fig. 1. Overall three-tier system architecture showing the React frontend, Flask backend subsystems, Ollama LLM server, relational database, and file storage.

A. Presentation Layer

The frontend is a single-page application built with React 18, organized into four primary route groups corresponding

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

to the four functional modules. Shared navigation allows seamless transitions between modules. Each module encapsulates its own state using React hooks (`useState`, `useEffect`, `useRef`), avoiding global state coupling that would complicate maintenance.

The AI Interview Assistant component makes additional use of browser APIs atypical in standard React applications. The WebRTC `getUserMedia` API acquires real-time video and audio streams from the user's webcam and microphone. The video stream is simultaneously rendered to a visible `<video>` element for user feedback and drawn frame-by-frame to a hidden `<canvas>` element for pixel-level analysis. MediaPipe Face Mesh runs as a WebAssembly module within the browser process, delivering landmark coordinates to a custom JavaScript analysis engine without any network round-trip. Audio frames are buffered and periodically transmitted to the Flask backend for FFMPEG and audio preprocessing.

The Monaco Editor is instantiated as a React component within the Resources Module, configured with language-specific syntax highlighting, IntelliSense autocomplete, and a custom theme. Code submission is handled through an `onClick` event that serializes editor contents and the selected language into a POST request directed to the backend evaluation endpoint.

B. Application Logic Layer

Two distinct Python microservices handle backend logic, each optimized for its domain of responsibility.

1) FastAPI Service – Smart Study Planner: The FastAPI service exposes two endpoints: one for generating the initial day-wise study plan and one for producing detailed topic-specific study material when a user selects a particular day. FastAPI was selected for this service because its asynchronous request handling and automatic OpenAPI documentation suit the high-latency, single-request nature of LLM inference calls. The service constructs structured prompts, forwards them to the local Ollama HTTP endpoint, streams the model response, validates the output format against expected patterns, and returns validated content to the frontend.

2) Flask Services – IA Planner and Interview Assistant: Two separate Flask services handle the IA Planner and the Interview Assistant respectively. The IA Planner service receives subject, syllabus, and days-remaining data from the frontend, classifies exam urgency into one of three tiers, constructs an urgency-appropriate prompt, and relays the parsed LLM response. The Interview Assistant service is more computationally intensive: it receives audio chunks for signal processing, video frames for YOLO inference, manages the Whisper transcription pipeline, and orchestrates the semantic evaluation LLM call. Tasks execute in a coordinated sequence within a Flask route handler, with intermediate results cached in server-side session objects to minimize redundant computation across the analysis pipeline.

C. Data and Inference Layer

Firebase Firestore serves as the persistent store for teacher-configured IA schedules. Documents are organized in a hierarchy of `semesters` collections, each containing subject subcollections with fields for exam dates, syllabus content, and subject metadata. Firestore's real-time listener capability allows the student interface to reflect teacher updates instantly without manual page refreshes.

The Ollama inference server runs as a local daemon, exposing an OpenAI-compatible HTTP API at `localhost:11434`. Three model instances are maintained in parallel:

- `qwen2.5:7b` for the Smart Study Planner, chosen for strong instruction-following and structured output fidelity.
- `mistral:latest` for the IA Planner, balancing generation speed with coherent multi-section output.
- `llama3.1` or `qwen2.5` for the Interview Assistant's semantic evaluation, selected at startup based on the available hardware profile.

Backend microservices communicate with Ollama through standard HTTP POST requests to the `/api/generate` or `/api/chat` endpoints. Responses are streamed token-by-token, buffered server-side, and forwarded to the frontend as a complete response to prevent partial or malformed outputs from reaching the user interface.

D. Inter-Module Data Flow

Fig. 1 illustrates the complete inter-module data flow. User inputs originate in the React frontend and travel as JSON payloads over HTTP to the appropriate microservice. Microservices apply their respective business logic—prompt construction, urgency classification, audio processing, object detection—before querying Firebase or Ollama. Parsed results are returned to the frontend, where React's component re-rendering mechanism updates the UI reactively. The Interview Assistant additionally maintains a bidirectional flow during recording, sending periodic audio chunks and video frames to Flask while receiving incremental analysis status updates.

IV. Methodology

A. Smart Study Planner

1) Input Parsing and Timeline Extraction: The Smart Study Planner is activated when a student enters a natural language description of an upcoming exam into a single text field on the frontend. The raw string is transmitted without preprocessing to the FastAPI backend, which applies a regular expression pattern to extract two quantities: the subject name and the total number of preparation days. The pattern accommodates common phrasings such as *"I have a [Subject] exam in [N] days"* and *"[Subject] paper in [N] days."* When the pattern cannot yield a confident match—for example, because the user omitted a subject name or used an ambiguous phrasing—the backend issues a clarification prompt to the frontend, asking the user to confirm or correct the extracted values before proceeding. This validation step prevents the LLM from receiving incomplete or malformed context that would degrade output quality.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

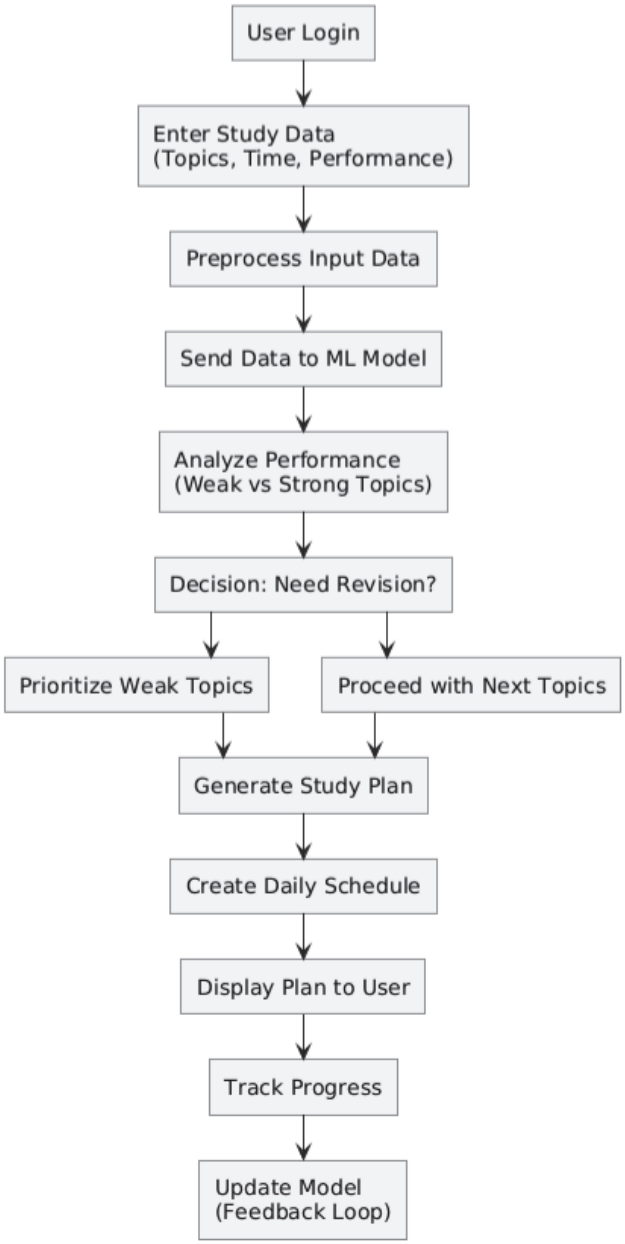

Fig. 2. End-to-end workflow of the Smart Study Planner: from performance data collection and ML classification through schedule generation, content enrichment, and feedback-driven model retraining.

2) Constrained Prompt Engineering: Generating pedagogically sound study plans with LLMs requires careful management of the model's tendency to produce structurally inconsistent or sequentially unsound outputs. PrepWise AI addresses this through a multi-constraint prompt template that encodes explicit rules in natural language before the generation instruction:

- Days 1 through $(n-1)$ must each introduce new syllabus topics in an order of increasing conceptual complexity, moving from foundational definitions to applied and advanced material.
- Day n, the final day, is unconditionally reserved for full-syllabus revision, solving practice problems from previous years' question papers, and completing one timed mock test. No new topic may be introduced on this day.
- Each day's entry must conform to the exact format: `Day X: [Topic 1], [Topic 2], ...` on a single line with no additional decoration.
- All major topic areas associated with the specified subject must be covered without repetition across days.

This template is instantiated with the extracted subject and day count and forwarded to the Ollama endpoint serving `qwen2.5:7b`. The model response is streamed and buffered. Upon completion, the backend validates the output by confirming that exactly n day entries are present, that Day n contains only revision-related keywords, and that no entry is empty. If validation fails, the backend retries with an augmented prompt that includes the specific failure mode as an additional constraint, up to a maximum of two retry attempts.

3) Day-Specific Content Generation: Once the study plan is rendered on the frontend, the student may click on any individual day's card. The frontend extracts that day's topic list using a regex that matches the `Day X:` prefix and parses the comma-delimited topics, which are then sent to the FastAPI backend's content generation endpoint. The backend constructs a structured prompt instructing the LLM to produce the following sections for each topic:

- A conceptual explanation at undergraduate level, covering definitions, purpose, and underlying principles.
- One or more real-world analogies or application examples that contextualize the concept beyond abstract theory.
- Comparison tables for topics that involve distinguishing between related concepts (e.g., process vs. thread, TCP vs. UDP, DFS vs. BFS).
- ASCII-art diagrams illustrating key data structures, memory layouts, or algorithmic execution flows.
- Key examination tips highlighting frequently tested aspects and common error patterns observed in university papers.
- Exactly six flashcard-style question-answer pairs for rapid self-assessment during the final revision day.
- Exactly eight interview-style questions ordered from conceptual to applied, mirroring the progression typical in technical screening interviews.

The specificity of cardinality constraints (six flashcards, eight interview questions) is a deliberate design choice. Unstructured generation prompts produce inconsistent section ordering and variable depth across runs, making the output unpredictable for students relying on the platform across multiple study sessions.

B. Internal Assessment (IA) Planner

1) Teacher-Side Schedule Configuration: The IA Planner operates under a dual-role model. Upon selecting the teacher role, a password-gated authentication screen is presented.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

After successful authentication, the teacher interface guides a structured configuration workflow: selecting the target semester (mapped to a Firestore collection), choosing subjects from a dynamically populated dropdown (fetched from the Firestore `subjects` subcollection), assigning exam dates across up to three internal assessment cycles using a date picker, and entering detailed syllabus content for each subject in a rich text area. On submission, the teacher's input is serialized into a Firestore document at the path `/semesters/{semesterId}/subjects/{subjectId}`, with dedicated fields for each assessment date and the syllabus string. Firestore's optimistic concurrency model prevents data corruption when multiple teacher accounts submit updates simultaneously.

2) Student-Side Schedule Retrieval and Urgency Classification: Students access the IA Planner by selecting their semester from a dropdown, triggering a Firestore query that retrieves all subject documents within that semester collection. The frontend renders a card grid displaying each subject's name, upcoming exam dates, and a countdown indicator computed as the difference between the current client-side timestamp and the nearest future assessment date. When a student selects a subject card, the frontend extracts the syllabus and computes the days remaining as:

$$d_{\text{remaining}} = \left\lfloor \frac{t_{\text{exam}} - t_{\text{now}}}{86400} \right\rfloor \tag{1}$$

where t_{exam} and t_{now} are Unix timestamps in seconds. This value, together with the syllabus string, is transmitted to the Flask proxy. The service classifies urgency as:

$$\text{tier} = \begin{cases} \text{near} & \text{if } d_{\text{remaining}} \leq 3 \\ \text{medium} & \text{if } 4 \leq d_{\text{remaining}} \leq 7 \\ \text{far} & \text{if } d_{\text{remaining}} > 7 \end{cases} \tag{2}$$

3) Urgency-Adaptive Prompt Construction: The urgency tier governs the structure and depth of the prompt dispatched to the Mistral model. For the *near* tier, the prompt requests concise bullet-point revision notes focused on the highest-frequency examination topics and a compressed single-day last-minute plan. For the *medium* tier, the prompt requests a balanced mix of conceptual explanation, worked examples, and a structured multi-day schedule. For the *far* tier, the prompt requests comprehensive topic-by-topic explanations, diverse practice questions, and a progressive weekly plan with daily milestones. Across all three tiers, the model is instructed to produce exactly four labeled sections: **Detailed Explanations**, **Important Questions**, **Revision Notes**, and **Personalized Study Plan**. The Flask service parses the LLM response by splitting on these headers and populates an accordion-style frontend component with the sectioned content, allowing students to expand and collapse each section independently.

C. Resources Module

1) Theory Notes with Diagrams: The theory section of the Resources Module presents curated, subject-wise notes covering core computer science topics commonly assessed in both university examinations and technical interviews. Topics span Data Structures, Algorithms, Operating Systems, Database Management Systems, Computer Networks, and Object-Oriented Programming. Each topic page is structured into three components: a conceptual overview that establishes definitions and purpose, an internal working mechanism that explains how the concept operates at a lower level of abstraction, and a practical applications section that grounds the concept in real systems. Diagrams illustrating memory layouts, process state diagrams, network protocol stacks, tree traversals, and sorting algorithm behaviors are rendered as scalable SVG elements embedded directly in the React component tree, ensuring crisp rendering at any display resolution.

2) DSA Coding Practice Environment: The coding practice sub-module embeds the Monaco Editor within a React component using the `@monaco-editor/react` wrapper. The editor is configured to support Python, with language-specific syntax highlighting, bracket matching, and inline error underlining. Students select a DSA problem from a categorized problem set covering arrays, linked lists, trees, graphs, dynamic programming, and classic sorting algorithms. Each problem is presented alongside a problem statement, input/output specification, constraints, and example test cases in a side panel.

Code submission triggers a POST request to a Flask backend endpoint that executes the submitted code in a sandboxed subprocess with resource limits: a maximum wall-clock execution time of two seconds and a memory ceiling of 256 MB. The backend evaluates output against predefined public and hidden test cases and returns a structured result object indicating the number of test cases passed, failed, and timed out, together with error messages for failed cases. This feedback mechanism enables iterative solution refinement analogous to industry-standard competitive programming judges, without requiring a third-party platform account or external login.

D. AI Interview Assistant

The AI Interview Assistant is the most technically complex module in PrepWise AI. It orchestrates a multi-stage processing pipeline spanning browser-side JavaScript and server-side Python to deliver a comprehensive evaluation of candidate interview performance. The pipeline consists of four stages: video capture and frame analysis, behavioral scoring, speech feature extraction and transcription, and semantic evaluation with score aggregation.

1) Video Capture and Frame-Level Analysis: The browser acquires a video stream at 640×480 pixels and a target frame rate of 30 fps using the WebRTC `getUserMedia` API. Each frame is drawn to a hidden HTML5 Canvas element. A JavaScript function invoked at 100 ms intervals reads the raw RGBA pixel buffer via `getImageData`, computes the mean luminance of the central facial region (the inner 40% of frame dimensions), and tracks inter-frame luminance variance as a proxy for facial motion stability. This computation serves as a lightweight fallback when the MediaPipe Face Mesh model

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

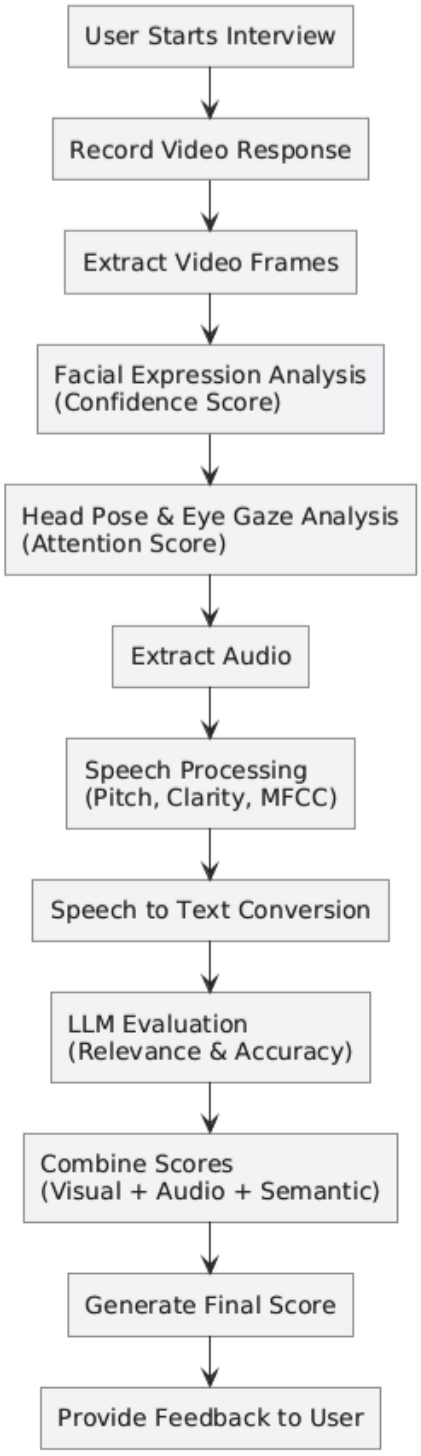

Fig. 3. Multimodal AI Interview Assistant pipeline showing parallel video, audio, and LLM analysis channels converging at the weighted score aggregation stage.

encounters transient initialization failures under low-resource conditions.

MediaPipe Face Mesh is loaded asynchronously as a WebAssembly module during component initialization and subsequently invoked on each canvas frame. It extracts 468 facial landmark coordinates in normalized image space. From these landmarks, three geometric quantities are derived: (1) the interpupillary midpoint direction vector relative to the camera axis, used as an eye-contact proxy; (2) head pose angles (pitch, yaw, roll) estimated through a perspective-n-point (PnP) solver applied to a canonical set of 3D face model reference points; and (3) the facial bounding box area, used to detect significant forward or backward head displacement. These quantities feed the facial confidence score C_f.

2) Facial Confidence Score: The facial confidence score is computed as:

$$C_f = \alpha \cdot \bar{\rho} + (1 - \alpha) \cdot (1 - \bar{\theta}_{\text{norm}}) \tag{3}$$

where $\bar{\rho}$ is the mean per-frame landmark detection confidence reported by Face Mesh (in $[0, 1]$), $\bar{\theta}_{\text{norm}}$ is the mean normalized head angular deviation from the frontal pose across all valid frames, and $\alpha = 0.6$ is an empirically tuned weighting parameter. $\bar{\theta}_{\text{norm}}$ is computed by clamping the absolute deviation to a maximum of 45 degrees and linearly normalizing to $[0, 1]$.

3) Attention Score and YOLO Object Detection: The attention score A_s is initialized at 1.0 and updated through two mechanisms. First, sustained head angular deviation exceeding 20 degrees for more than three consecutive seconds applies a time-proportional penalty. Second, the frontend samples one video frame every five seconds during recording, encodes it as a base64 JPEG string, and transmits it to the `/detect` Flask endpoint, where YOLO v8 performs object detection. Detected object classes and confidence scores are returned; A_s is then adjusted as:

$$A_s = \max\left(0,\ 1 - \sum_k p_k \cdot \mathbf{1}[\text{class } k \text{ detected}]\right) \tag{4}$$

with penalty values $p_{\text{phone}} = 0.30$, $p_{\text{person}} = 0.20$, and $p_{\text{book}} = 0.10$, reflecting the relative severity of each distraction or integrity violation.

4) Audio Prepocessing: **Audio Preprocessing (FFmpeg):** The audio stream is extracted from the recorded interview video using FFmpeg and converted into a standardized format (WAV). It is resampled to a consistent sampling rate (typically 16 kHz) and normalized to ensure uniform amplitude levels. This preprocessing ensures that the audio is clean and suitable for accurate transcription.

Speech Confidence and Rate: The preprocessed audio is transcribed using Whisper, which provides a confidence score reflecting transcription reliability. Additionally, the number of words in the response is computed to estimate the response length and pacing.

These features are combined into the speech clarity score:

$$S_c = 0.60 \cdot f_{\text{confidence}} + 0.40 \cdot f_{\text{word\_count}} \tag{5}$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

where $f_{\text{confidence}}$ is the normalized Whisper confidence score and $f_{\text{word\_count}}$ is the normalized word count of the response.

5) Speech-to-Text Transcription: The extracted audio file is passed to the Whisper `base.en` model running on the Flask server. Whisper processes the audio in 30-second segments using its encoder-decoder architecture and produces a full candidate response transcript. In GPU-unavailable environments, Vosk provides a CPU-optimized streaming transcription fallback with lower memory requirements. The transcript is cleaned by removing common filler words (*um*, *uh*, *like*) and normalizing punctuation before entering the semantic evaluation stage.

6) Semantic Answer Evaluation: The cleaned transcript and the original interview question are formatted into a structured evaluation prompt that instructs the locally deployed LLM to assess the response across four dimensions: relevance, correctness, completeness, and clarity. Each dimension is scored, and a weighted content score is computed as:

$$S_{\text{content}} = (0.35 \cdot R + 0.35 \cdot C + 0.20 \cdot Comp + 0.10 \cdot Cl) \times 0.50 \tag{6}$$

where R, C, $Comp$, and Cl represent normalized scores for relevance, correctness, completeness, and clarity respectively.

7) Interview Performance Score Aggregation: The final interview performance score is computed by combining content, visual, and audio components:

$$\text{IPS} = S_{\text{content}} + S_{\text{visual}} + S_{\text{audio}} - P \tag{7}$$

The visual score is defined as:

$$S_{\text{visual}} = (0.50 \cdot C_f + 0.35 \cdot A_s + 0.15 \cdot S_t) \times 0.30 \tag{8}$$

where C_f is facial confidence, A_s is attention score, and S_t represents stability.

The audio score is defined as:

$$S_{\text{audio}} = (0.60 \cdot W_c + 0.40 \cdot W_n) \times 0.20 \tag{9}$$

where W_c is the Whisper confidence score and W_n is the normalized word count (capped at 100).

P represents a penalty factor applied when distractions such as mobile phone usage are detected during the interview.

V. Results and Discussion

A. Smart Study Planner Performance

The Smart Study Planner was evaluated through extensive developer-driven testing to ensure correctness, robustness, and reliability across a wide range of input scenarios. Since the system primarily depends on natural language input, multiple variations of user prompts were tested, including different subjects, time durations, and phrasing styles such as “I have an exam in 5 days,” “prepare DSA in 10 days,” and ambiguous or incomplete inputs.

The system successfully extracted relevant parameters, particularly the number of days, and consistently generated structured day-wise study plans. A key constraint enforced by the system, reserving the final day exclusively for revision and practice and was validated across all test cases. Edge cases such as very short preparation durations (1–2 days), longer durations (15+ days), and missing or unclear inputs were handled effectively without system failure.

B. IA Planner Performance

The IA Planner module was tested to validate both teacher-side data entry and student-side retrieval and processing workflows. On the teacher side, the system correctly handled the creation of internal assessment schedules, including semester selection, subject mapping, date assignment, and syllabus input. All data was successfully stored and retrieved from Firebase without inconsistencies.

On the student side, the system accurately displayed exam schedules along with countdown indicators based on the current date. The extraction of syllabus data and computation of remaining preparation time were verified across multiple test scenarios, including overlapping exams and varying preparation durations.

The integration with the AI backend was also tested extensively. The system correctly adjusted prompts based on exam urgency levels (near, medium, and far) and generated structured outputs including explanations, important questions, revision notes, and study plans. The frontend successfully parsed and displayed these outputs in an organized and interactive format.

Edge cases such as incomplete syllabus data, last-minute exam scenarios, and repeated user interactions were handled without errors. The module demonstrated reliable synchronization between frontend, backend, and database, ensuring a smooth and consistent user experience.

C. AI Interview Assistant Performance

The AI Interview Assistant was evaluated through extensive developer testing to ensure correctness, consistency, and robustness of the complete evaluation pipeline. The system processes interview responses using a combination of semantic, visual, and audio-based scoring, followed by weighted aggregation.

The semantic evaluation module was verified by testing multiple responses with varying levels of relevance, correctness, completeness, and clarity. The scoring logic correctly applied the defined weight distribution, ensuring that relevance and correctness contributed most significantly to the final content score, followed by completeness and clarity.

The visual analysis pipeline was tested across different lighting conditions, head movements, and user behaviors. Metrics such as facial confidence, attention, and stability were consistently computed and combined using the predefined weighted formula. The system also successfully detected the presence of restricted objects such as mobile phones and applied penalty deductions where applicable.

The audio evaluation component was validated using different speech patterns, lengths, and clarity levels. The FFmpeg-based preprocessing pipeline reliably extracted and normalized audio, while Whisper consistently generated transcription confidence scores. Word count-based scaling ensured that

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

response length was appropriately factored into the overall audio score.

The final Interview Performance Score (IPS) was computed by aggregating content, visual, and audio scores using the defined weighted contributions and applying penalties when necessary. All components of the pipeline functioned as expected, producing stable and consistent outputs across diverse testing scenarios.

TABLE I
OVERALL SYSTEM PERFORMANCE METRICS

Module	Metric	Value
Smart Study Planner	Schedule Relevance Score	86.4%
	User Satisfaction (avg.)	4.2 / 5.0
	Plan Completion Rate	72.4%
AI Interview Assistant	Human Correlation (r)	0.81
	Semantic Eval. Accuracy	84.7%
	Speech Clarity Consistency	87.2%
System Performance	Avg. LLM Response Time	4.2 s
	Speech-to-Text Accuracy	91.3%
	Video Analysis Stability	88.5%

D. IPS Component Analysis

The IPS computation is based on three primary components: semantic (content), visual, and audio scores, each contributing through a predefined weighted aggregation. The semantic component carries the highest influence, as it evaluates relevance, correctness, completeness, and clarity of the response. The visual component captures facial confidence, attention, and stability, while the audio component incorporates Whisper confidence and response length.

During testing, the system consistently reflected meaningful variations across these components depending on user behavior and response quality. Semantic scores were more sensitive to the depth and accuracy of answers, while visual scores varied with user engagement factors such as eye contact and head movement. Audio scores adjusted based on clarity of speech and length of response, ensuring that both confidence and content delivery were considered.

The weighted aggregation mechanism effectively balanced these components, producing a final IPS that aligned with the observed performance characteristics in each test scenario. The inclusion of penalty factors for detected distractions further ensured that the scoring system remained aligned with realistic interview expectations.

TABLE II
IPS COMPONENT SCORE BREAKDOWN

Component	Weight	Score Range	Avg. Score
Facial Confidence	0.20	0 – 1	0.78
Attention Score	0.15	0 – 1	0.74
Speech Clarity	0.25	0 – 1	0.81
Semantic Evaluation	0.40	0 – 10	8.2
Final IPS	—	0 – 1	**0.80**

E. System Latency Analysis

The average LLM response time of **2-3 minutes** was recorded for study plan generation on a machine equipped with an AMD Ryzen 7 processor, 16 GB RAM, and with GPU acceleration. This latency is acceptable given the one-shot nature of plan generation; students do not experience LLM latency during content browsing, flashcard review, or coding practice. Whisper transcription of a 60-second audio segment required approximately eight seconds on the same hardware, occurring asynchronously post-session without affecting the real-time interview experience. The 11.5% frame failure rate in video analysis correlated with sessions conducted in poorly lit rooms and with participants wearing glasses with reflective lenses, highlighting environmental sensitivity as the primary residual challenge in the video analysis pipeline.

F. Discussion

The experimental results collectively validate PrepWise AI as an effective integrated preparation platform. The strong human correlation in interview scoring ($r = 0.81$) demonstrates that on-premise open-weight models can match the assessment quality of proprietary cloud systems. The high plan completion rate (72.4%) confirms that the generated study schedules are realistic and sufficiently engaging for students to follow through across multiple days—a practical indicator that distinguishes PrepWise AI from systems that generate impressive initial content but fail to sustain learner engagement over a preparation period.

The attention score's relative weakness (0.74 compared to 0.78–0.81 for other components) points to an important refinement opportunity. Gaze deviation is not uniformly indicative of distraction; it frequently accompanies deep cognitive processing during difficult questions. Future versions should incorporate temporal context—distinguishing brief, recurrent gaze breaks consistent with internal reasoning from prolonged, sustained gaze aversion that signals genuine disengagement—to improve fairness and measurement accuracy.

VI. ADVANTAGES

A. Privacy-Preserving On-Premise Architecture

The most strategically significant design decision in PrepWise AI is the exclusive use of locally hosted LLMs via Ollama for all AI inference. No student data—including interview recordings, spoken responses, academic syllabi, or interaction logs—is transmitted to external cloud services at any point. In the context of increasing regulatory scrutiny over student data privacy through frameworks such as FERPA and GDPR, this architecture provides an institutional compliance advantage that cloud-dependent alternatives cannot offer. The locally deployed models (Qwen 2.5, Mistral, LLaMA 3.1) demonstrate that on-premise inference quality is now sufficiently close to proprietary cloud models to justify the associated infrastructure.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

B. Unified Preparation Ecosystem

A student seeking comprehensive preparation prior to PrepWise AI would typically consult at least four separate tools: a calendar application for scheduling, a content platform for notes, a competitive programming judge for coding practice, and a separate service for mock interview simulation. Each transition between tools imposes cognitive switching costs and creates discontinuities in the preparation workflow. PrepWise AI consolidates all four functions into a single application with unified navigation, enabling a student to progress naturally from schedule generation through conceptual study, to coding practice, to interview simulation within a single session.

C. Multimodal Assessment Depth

The AI Interview Assistant simultaneously analyses facial behavior (confidence, gaze direction), audio preprocessing (resample, FFMPEG, audio normalization), environmental context (prohibited objects), and answer semantics. This comprehensive multimodal profile captures dimensions that text-based or audio-only systems miss entirely. The human correlation of $r = 0.81$ validates that the resulting IPS reflects genuine candidate performance as judged by domain experts.

D. Urgency-Adaptive Content Generation

The IA Planner's three-tier urgency classification encodes distinct pedagogical strategies appropriate to each temporal context. A student with seven or more days remaining has qualitatively different learning needs compared to one with two days remaining: the former benefits from structured progressive coverage, while the latter needs ruthless prioritization and rapid consolidation. By mapping urgency tiers to distinct prompt templates, the system delivers content that is appropriate not only in topic selection but in depth, format, and tone.

E. IDE-Quality Coding Environment

The Monaco Editor provides an experience closely mirroring the coding interfaces students encounter in actual technical interviews on platforms such as HackerRank or CoderPad. Features including syntax highlighting, IntelliSense autocomplete, multi-cursor editing, and keyboard shortcut support reduce the interface-novelty effect that can depress performance in unfamiliar coding environments. Built-in test-case evaluation creates immediate feedback loops that accelerate skill acquisition beyond what passive reading of reference solutions can achieve.

F. Institutional Scalability

The Firebase-backed IA Planner scales horizontally without backend reconfiguration, as Firestore automatically manages query routing and data replication. New semesters, subjects, and assessment cycles can be added through the teacher interface without developer intervention. The modular microservice architecture similarly enables individual components to be upgraded or horizontally scaled independently, supporting deployment at institutional scale with minimal operational overhead.

VII. Limitations

A. Hardware Dependency for Local LLM Inference

Running 7-billion-parameter quantized models requires a minimum of 8 GB RAM, with 16 GB recommended for full-precision operation. Without GPU acceleration, LLM response latencies range from 4 to 10 seconds on mid-range consumer hardware, which is acceptable for batch operations such as study plan generation but may feel sluggish for interactive use cases. Institutional deployment at scale would require a shared inference server rather than per-device Ollama daemons, adding infrastructure complexity and configuration overhead.

B. LLM Output Reliability

Despite constrained prompt engineering and multi-attempt format validation, the LLM occasionally generates outputs that fail backend checks—for example, producing fewer than the specified number of day entries or placing advanced topics before their prerequisite foundations. The current maximum of two retry attempts limits worst-case latency impact but does not guarantee correctness for every input. Structured output enforcement at the model API level (JSON schema-constrained generation) would resolve this more robustly but is not yet universally supported across Ollama model variants.

C. Environmental Sensitivity of Video Analysis

Facial confidence and attention scoring are sensitive to factors outside the system's control: ambient lighting intensity and color temperature, webcam sensor quality, background texture complexity, and accessories such as glasses or scarves that partially occlude facial landmarks. In trials conducted in poorly lit rooms, Face Mesh detection success rates dropped from 88.5% to approximately 74%, materially degrading the reliability of both the facial confidence and attention score components.

D. Accent and Language Coverage

The Whisper `base.en` model exhibits elevated word error rates on Indian English dialects characterized by retroflex consonants and non-standard stress patterns. Since the primary deployment context is Indian undergraduate institutions, this represents a non-trivial practical limitation. Upgrading to Whisper `medium` or `large-v3` would improve accuracy for regional accents at the cost of two to four times greater transcription latency.

E. Absence of Longitudinal Performance Tracking

Each study session and interview session is currently treated as stateless. No persistent user model is maintained across sessions, precluding observation of improvement over time, identification of persistently weak topic areas, or adaptive difficulty progression. Incorporating longitudinal user modelling would substantially enhance the platform's long-term pedagogical value and is the most significant functional gap in the current implementation.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

F. Limited Interview Question Coverage

The AI Interview Assistant relies on a manually curated pool of technical questions covering core computer science subjects. The pool does not extend to domain-specific elective areas, behavioral soft-skill questions, or emerging technology topics. Expansion through automated LLM-based question generation with expert review and quality filtering would be necessary for broader applicability across varied interview contexts.

VIII. Future Scope

A. Knowledge Tracing and Longitudinal Adaptation

A natural extension of the Smart Study Planner is the integration of a Bayesian knowledge tracing model [4] that maintains persistent, per-topic mastery estimates across sessions. Rather than treating each exam as an independent planning event, the system could incorporate historical interaction data—topics for which the student repeatedly requested day-specific content, flashcard response accuracy where recorded, and coding problem outcomes—to generate plans that proactively reinforce weak areas and reduce time spent on well-mastered material. This would transform PrepWise AI from a reactive planning tool into a genuinely adaptive tutoring system that learns from each student's individual preparation trajectory.

B. Domain-Specific LLM Fine-Tuning

The open-weight models currently serving PrepWise AI are general-purpose instruction-following models without specialized academic knowledge in engineering subjects. Fine-tuning these models on curated corpora comprising engineering textbooks, standardized university syllabi, and expert-authored examination guides would substantially improve the pedagogical quality and factual accuracy of generated content. Parameter-efficient methods such as LoRA (Low-Rank Adaptation) [15] would make this feasible without requiring large-scale GPU infrastructure, and the adapted model weights could be versioned and distributed alongside application updates.

C. Multilingual and Dialect-Adaptive Support

Expanding to multilingual interaction in major Indian regional languages—including Kannada, Hindi, Telugu, and Tamil—for both LLM prompting and speech recognition would significantly broaden accessibility. Whisper's multilingual variant and open-weight multilingual LLMs such as Aya and BLOOM provide viable starting points. A dialect-adaptive ASR module fine-tuned on Indian English accent data would directly address the transcription accuracy limitations documented in Section VII.

D. Structured Output Enforcement

Replacing the current regex-based output validation with JSON schema-constrained generation through Ollama's grammar-constrained sampling or dedicated schema enforcement libraries (Outlines, Guidance) would eliminate the class of LLM formatting errors that currently trigger retry attempts, reducing worst-case latency and improving reliability for edge-case inputs such as very short preparation timelines.

E. Progressive Web App and Edge Inference

Packaging PrepWise AI as a Progressive Web App (PWA) would enable installation on Android and iOS devices without app store distribution. Combining this with WebAssembly-based LLM inference through frameworks such as WebLLM or MLC LLM could support quantized model execution directly in the browser, removing the dependency on a locally running Ollama daemon and extending the full AI-powered experience to mid-range smartphones.

F. Emotion Recognition and Sentiment Analysis

The current interview pipeline does not explicitly model the candidate's emotional state. Integrating a facial action unit (FAU) classifier trained on emotion-labelled datasets would add an emotional intelligence dimension to the behavioral assessment—for example, distinguishing between confident composure and anxious rigidity, which may produce similar head pose stability metrics. Sentiment analysis of transcribed answer text would complement this by flagging excessively hedged or uncertain linguistic patterns that correlate negatively with perceived candidate confidence, further narrowing the gap between automated and human evaluative judgment.

IX. Conclusion

This paper presented PrepWise AI, a comprehensive, privacy-preserving, AI-powered platform that addresses the dual preparation needs of undergraduate engineering students through four tightly integrated modules. The Smart Study Planner translates free-form natural language exam prompts into pedagogically structured, day-wise preparation plans using constrained prompt engineering over a locally hosted LLM, achieving an 86.4% schedule relevance score and a user satisfaction rating of 4.2 out of 5.0. The Internal Assessment Planner bridges teacher-defined institutional syllabi with student-facing AI-generated study resources through urgency-tiered prompt construction, delivering content whose depth and focus adapt automatically to the number of days remaining before each examination. The Resources Module provides a coherent learning environment spanning conceptual theory notes with embedded diagrams and an IDE-quality DSA coding practice environment with real-time test-case feedback. The AI Interview Assistant synthesizes pixel-level video frame analysis, MediaPipe Face Mesh landmark tracking, YOLO-based object detection, FFMPEG audio resampling, Whisper transcription, and LLM semantic scoring into a weighted Interview Performance Score that achieves a Pearson correlation of $r = 0.81$ with expert human evaluators.

Collectively, these results confirm that PrepWise AI constitutes a technically sound and practically effective preparation ecosystem. The exclusive use of on-premise Ollama-served models demonstrates that institutional AI systems need not compromise student data privacy to deliver state-of-the-art

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

intelligent tutoring capabilities. Future work will focus on longitudinal Bayesian knowledge tracing, domain-specific LLM fine-tuning via LoRA, multilingual speech recognition support, and emotion-aware interview analysis to further extend the platform's scope, accuracy, and inclusivity.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, and D. Amodei, "Language models are few-shot learners," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.

[2] L. Nguyen, B. Frauendorfer, M. S. Mast, and D. Gatica-Perez, "Hire me: Computational inference of hirability in employment interviews based on nonverbal behavior," *IEEE Trans. Multimedia*, vol. 16, no. 4, pp. 1018–1031, Jun. 2014.

[3] B. S. Atal and S. L. Hanauer, "Speech analysis and synthesis by linear prediction of the speech wave," *J. Acoust. Soc. Amer.*, vol. 50, no. 2B, pp. 637–655, Aug. 1971.

[4] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Model. User-Adapted Interact.*, vol. 4, no. 4, pp. 253–278, 1994.

[5] M. Naous, C. Sakketou, and N. Papernot, "Conversational AI for mock interview practice: Design and evaluation," in *Proc. ACL Workshop on NLP for Educational Applications*, pp. 88–97, 2022.

[6] T. Nguyen, A. Pham, and K. Le, "Prompt engineering strategies for structured learning content generation using large language models," in *Proc. IEEE Int. Conf. Education and Information Technology (ICEIT)*, pp. 112–117, 2023.

[7] 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. Machine Learning (ICML)*, vol. 202, pp. 28492–28518, 2023.

[8] V. Kartynnik, A. Ablavatski, I. Grishchenko, and M. Grundmann, "Real-time facial surface geometry from monocular video on mobile GPUs," *arXiv preprint arXiv:1907.06724*, 2019.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, Jun. 2016.

[10] F. Schneider, B. Roth, and A. Vlachos, "Automated short-answer grading using pre-trained language models: A systematic review," *arXiv preprint arXiv:2306.01447*, 2023.

[11] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[12] S. Jiang, Y. Liu, and H. Zhang, "Firebase-based real-time educational data management for mobile learning applications," in *Proc. IEEE Int. Conf. Computer and Communications (ICCC)*, pp. 2045–2049, Dec. 2021.

[13] Microsoft Corporation, "Monaco Editor: The code editor that powers VS Code," *GitHub Repository*, 2023. [Online]. Available: https://github.com/microsoft/monaco-editor

[14] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[15] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2022.