JIRBDAI

Journal of Institutional Research, Big Data Analytics and Innovation

An Enhanced Web-Based Examination System using Automated Proctoring and Background Activity Detection

Toyyib Olaitan Olanrewaju[1], Rafiu Mope Isiaka[2], Seyi Ronke Babatunde[3], Muhammed Kamaldeen Jimoh [4], & Nathaniel Akinbowale Babatunde[5]

[1] Department of Computer Science, Faculty of Information and Technology, Kwara State University, Malete,
[2]Department of Computer Science, Faculty of Information and Technology, Kwara State University, Malete,
[3]Department of Computer Science, Faculty of Information and Technology, Kwara State University, Malete,
[4]Department of Computer Science, Faculty of Education, University of Ilorin, Ilorin, &
[5]Department of Computer Science, Faculty of Information and Technology, Kwara State University, Malete.

*Corresponding Authors: [1]toyyib.olanrewaju@kwasu.edu.ng,[2] abdulrafiu.isiaka@kwasu.edu.ng,[3] ronke.babatunde@kwasu.edu.ng, [4]jimoh.km@unilorin.edu.ng, & [5]akinbowale.babatunde@kwasu.edu.ng.

ABSTRACT The rise of online education demands sturdy systems to uphold academic integrity during remote assessments. This paper presents the design and implementation of a web-based examination platform integrated with automated, multi-modal proctoring tools to detect potential cheating. The system enables educators to create and manage diverse assessments while students' complete exams in a monitored virtual environment. Key security measures include user authentication with facial recognition using the DeepFace library and continuous webcam analysis to detect mobile phone usage, unauthorized individuals, and unusual head or eye movements. Additional features include browser focus tracking to monitor navigation away from the exam window, and background audio analysis using Voice Activity Detection (VAD) and Root Mean Square (RMS) energy to flag suspicious sounds or conversations. Detected anomalies are logged in real-time for review. The prototype effectively combines visual, auditory, and behavioral monitoring into a cohesive framework, offering a comprehensive solution to reduce academic misconduct in remote settings. Functional testing showed 72% overall system accuracy and over 90%+ accuracy in specific detection modules. Future work should intend to enhance the system's AI capabilities, improve performance on models, and ensure fairness for all users, contributing to credible and secure digital learning environments across disciplines.

KEYWORDS: Voice Activity Detection, Background Activity Detection, Server-Sent Events, Root Mean Square, Audio Cheating, You Only Look Once.

I. INTRODUCTION

Traditional examinations required students to be physically present in classrooms under the supervision of human invigilators to deter malpractice (Shilpa *et al.,* 2023). However, with the rapid growth of technology and the adoption of virtual learning platforms, online examinations have become common, creating new challenges in maintaining academic integrity. Automated proctoring systems have emerged to address these issues, but most focus primarily on visual monitoring such as multiple face detection, screen violation detection, and eye movement tracking (Tweissi *et al.,* 2022) while neglecting the detection of subtle audio cues. Voice Activity Detection (VAD), widely used in fields such as home care (Van Hengel & Anemüller, 2019), surveillance (Kotus *et al.,* 2019), environmental monitoring (Stowell *et al.,* 2020), and urban traffic control (Meucci *et al.,* 2020), offers potential for

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

enhancing online examination security. Yet, existing proctoring systems rarely incorporate it to detect background events like whispers, overlapping speech, or coded audio signals, which can threaten examination integrity if left unnoticed (Rahardjo *et al.,* 2024). Research indicates a need for solutions capable of assessing environmental audio conditions to determine whether an examination setting is conducive to fair assessment (V *et al.,* 2023; Amer-Yahia, 2022; Niveditha *et al.,* 2023). To bridge this gap, this study proposes a multimodal AI-based proctoring framework that integrates real-time VAD with conventional visual monitoring techniques. The system classifies the environment as conducive or disruptive based on detected noise, flagging suspicious sounds while maintaining a non-invasive design (Waleed *et al.,* 2023). This approach not only strengthens the reliability of automated proctoring but also contributes to a serene and fair examination environment. By embedding audio-based verification into multimodal monitoring, the research addresses an overlooked dimension of online exam security, aligning with the broader goal of promoting secure, adaptive, and trustworthy digital learning environments.

II. LITERATURE REVIEW

A. Overview of Proctoring Systems

Proctoring systems use digital technologies to ensure academic integrity in online examinations. These systems streamline exam delivery, reduce misconduct, and enable efficient student evaluation. Various approaches exist, including decision-support models like VIKOR (Dinda *et al.,* 2018), face recognition-enabled portals (Hemant *et al.,* 2022), and multi-modal biometric authentication systems such as CBUA-OE (Purohit *et al.,* 2022), all aimed at improving monitoring effectiveness and security.

B. In-Person vs Remote Proctoring

While traditional in-person proctoring has been widely used, technological advances and the COVID-19 pandemic have accelerated the adoption of remote proctoring. Studies comparing both methods reveal differences in fairness, data privacy, and user experience (Elshafey *et al.,* 2021; Harmon *et al.,* 2017). Automated tools, including face and head pose detection, now enhance online exam supervision (Potluri *et al.,* 2023), improving data quality and reducing careless behaviors (Francavilla *et al.,* 2019).

C. Live and Recorded Proctoring

Live online proctoring systems, such as the 'Attentive system' (Tejaswi *et al.,* 2023), use AI-based features like face spoofing detection and real-time behavior monitoring to uphold integrity during exams. Recorded proctoring methods, where sessions are reviewed after completion (Chun *et al.,* 2022), are effective for flagging suspicious behaviors without disrupting performance. These systems assess facial expressions, eye movements, and background noise for possible violations (Jia & He, 2022).

D. Automated AI Proctoring

AI-driven proctoring automates the entire monitoring process using technologies such as eye gaze tracking, lip movement analysis, and real-time alerts (Shilpa *et al.,* 2023). These systems improve security, reduce logistical burdens, and provide a user-friendly environment. Students log in, verify identity via facial input, undergo system checks, and are continuously monitored during the exam. Final scores combine performance and a trust score reflecting exam rule adherence.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

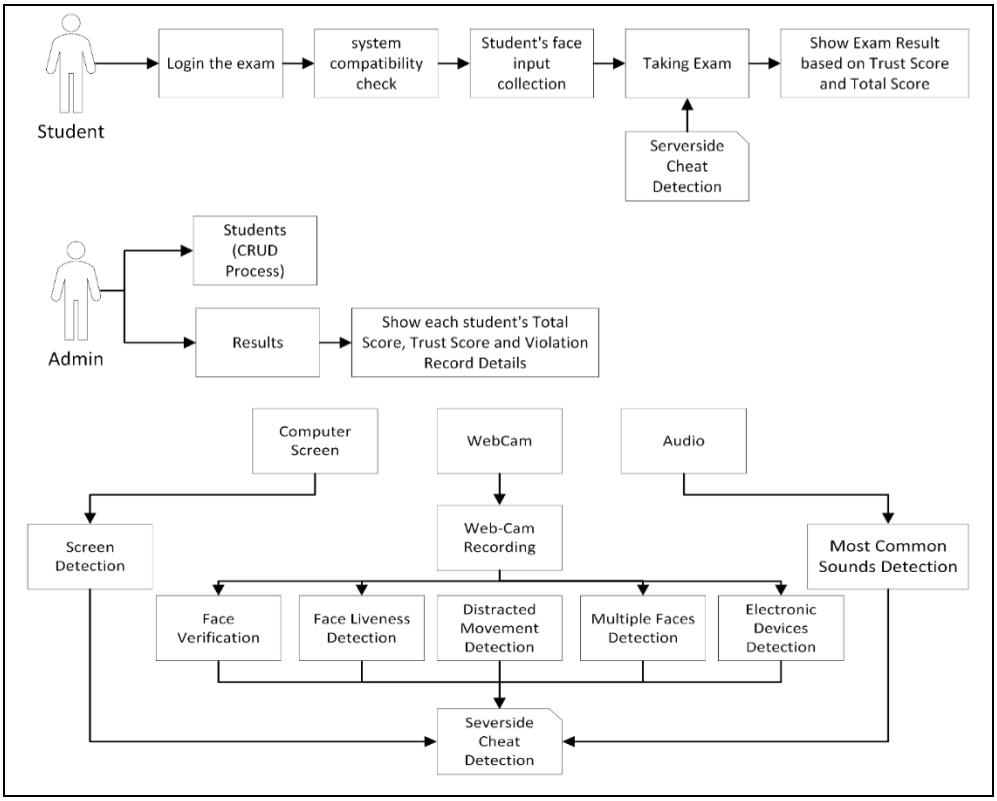

Figure 1. Interaction Flow in Automated Proctoring with Server-side Monitoring (Tweissi *et al.,* 2022).

E. Administration & Server-Side Operations

Administrators in automated proctoring systems can perform CRUD operations on student records and access detailed exam results, including trust scores and violation logs.

Server-side cheat detection features include screen monitoring to prevent tab switching, webcam recording with face verification and liveness detection, and audio analysis to flag suspicious sounds.

F. Hybrid & Mobile Proctoring

Hybrid proctoring combines biometric verification, facial recognition, and behavior tracking to enhance exam integrity across online and in-person settings (Purohit *et al.,* 2021).

Mobile proctoring, developed in response to the pandemic, enables students to scan handwritten responses and incorporates features like eye tracking and mobile phone detection (Yaghi *et al.,* 2022; Ranka *et al.,* 2023). These tools expand proctoring access and enable remote expert guidance in various fields.

G. Lockdown Browsers

Lockdown browsers enforce secure exam conditions by restricting access to unauthorized content. Developed during the COVID-19 lockdown, these tools also reflect shifting browsing behaviors and highlight digital security needs, especially among different gender groups (Miller *et al.,* 2023).

H. Theory & Broader Applications

Proctor systems have broad utility from online exams to medical training and data center management. They employ statistical and AI techniques for behavior monitoring, intrusion detection, and performance optimization (Kannan *et al.,* 2018; Murakami *et al.,* 2023).

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

I. Computer-Aided Instruction

In PSI (Personalized System of Instruction), proctor systems like "Tester" and "Editor" modules allow flexible, self-paced learning with automated quiz management and record editing. These tools streamline course management and adapt content delivery (Charles *et al.,* 2022).

J. Privacy Protection & ECOVAD

ecoVAD is a neural model for detecting human speech in natural environments. It ensures privacy while enabling large-scale ecological monitoring, outperforming standard voice detection models with high F1 scores and accurate speech detection up to 20 meters.

K. Neural Vocoders & Synthetic Voice Detection

Neural vocoders (autoregressive, GAN, and diffusion-based) improve speech synthesis. New detection frameworks can now identify synthetic voices using vocoder-specific artifacts, addressing misuse in VAD systems (Wu *et al.,* 2015; Todisco *et al.,* 2019).

L. Emotion Detection from Voice

A real-time emotion detection system using *BoAW* and Bi-RNN with attention accurately classifies emotions in conversations. It extracts audio features with *openSMILE* and models temporal context, enabling robust audio emotion recognition.

M. Digital Signal Processing (DSP)

DSP systems use digital filters (FIR, IIR) to remove noise from audio. Filter types are chosen based on noise characteristics low-pass, high-pass, bandpass, and notch filters to enhance signal clarity during audio monitoring.

N. Voice Activity Detection (VAD)

VAD distinguishes speech from noise to improve speech system efficiency. Traditional methods rely on energy thresholds, while modern approaches use CRNNs and unsupervised models like K-Means. Advanced techniques now focus on robustness in diverse environments, using features like harmonicity, modulation, and deep learning (Graf *et al.,* 2015; Patil & Patil, 2024).

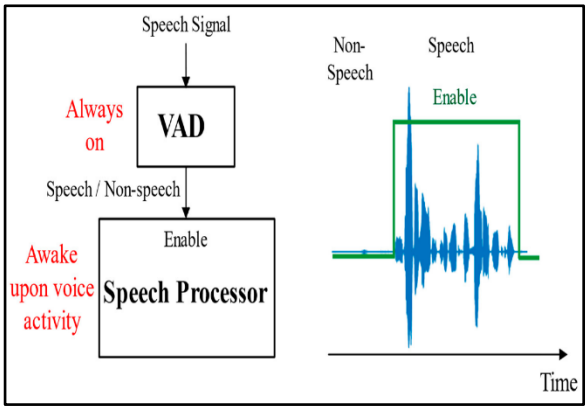

Figure 2. Always-on Voice Activity Detection as a Wakeup Mechanism *(*Faghani *et al., 2023).*

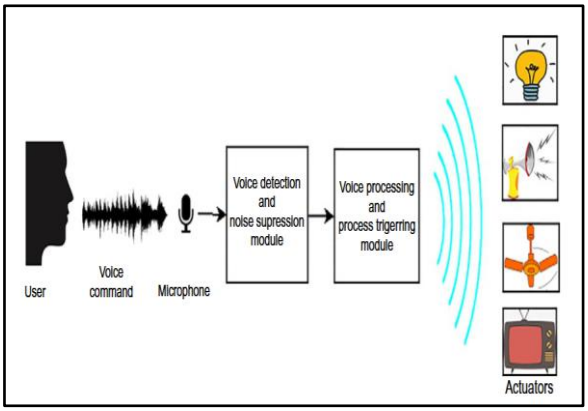

Figure 3. Conceptual Design of the VAD-based Automation System *(*Jat *et al., 2019).*

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

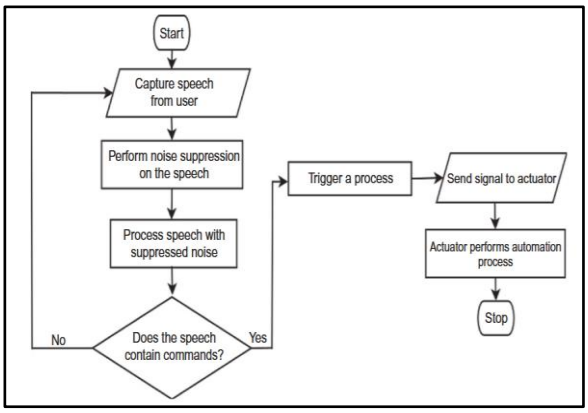

Figure 4. Flowchart of the VAD-based Automation System *(Jat et al., 2019).*

O. Related Work

In recent years, several researchers have proposed intelligent systems intended for enhancing the integrity and efficiency of online examinations through automated proctoring solutions. *Hemant et al.* (2022) developed an online examination portal titled "Exam Conduction and Proctoring System Using Face Detection" to simplify the process of exam creation, invigilation, paper evaluation, and performance assessment. The portal was designed using a technology stack comprising *HTML, CSS*, *Bootstrap, JavaScript, Node.js, Express.js*, and *MongoDB*, with Face *API* integrated for facial recognition. Although the system demonstrated functional efficacy, it lacked resilience during network or power disruptions, suggesting the need for future integration of intelligent fault-tolerant mechanisms. Similarly, Fatima *et al.* (2022) introduced an intelligent exam supervision system powered by deep learning techniques, specifically aiming to mitigate cheating in real-time assessments. Leveraging Faster Regional Convolutional Neural Networks (RCNN) and Multi-task Cascaded Convolutional Neural Networks (MTCNN), their system achieved high training and testing accuracies of 99.5% and 98.5%, respectively. The model effectively detected suspicious student activities such as head movement and face recognition, with the capability to monitor over 100 students simultaneously. However, the authors suggested that future improvements should include the detection of hand gestures and the identification of prohibited items through additional object detection models such as *YOLOv4, RCNN,* and Mask *RCNN.*

In a similar vein, Shkodzinsky and Lutskiv (2022) presented an AI-driven proctoring system integrated into the *LMS* A Tutor. Their work focused on enhancing identity verification by deploying face detection and recognition algorithms such as HOG+SVM and CNN. This system achieved a notable accuracy of 96% under varying lighting conditions and even when candidates wore face masks. Deployed at Ternopil National Technical University (TNTU), it demonstrated moderate false acceptance (0.75%) and false rejection (3.25%) rates. Despite minor limitations caused by environmental lighting, the system provided consistent proctoring support, ultimately deferring final decisions to human invigilators. Expanding the domain of proctoring technologies, Tejaswi *et al.* (2023) introduced the Attentive System, an automated proctoring solution built on attentive-net technology. This system was designed to identify cheating behaviors by incorporating face detection, face spoofing detection, multiple person detection, and head pose estimation. Validated using the Crime Investigation and Prevention Lab (*CIPL*) dataset, the system achieved an accuracy of 0.87. While effective in detecting misaligned or occluded faces, the authors acknowledged limitations related to neutral expressions and partial visibility, recommending future enhancements such as expanded training data and audio-capturing capabilities.

In parallel to vision-based monitoring systems, audio analysis has emerged as a complementary approach in intelligent surveillance. Ioannis et al. (2020) explored the use of two-dimensional spectrogram magnitude representations to improve event detection in noisy environments, particularly in autonomous vehicles (*AVs*). Their

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

study evaluated multiple features, including Short-Time Fourier Transform *(STFT)*, Mel-scale, and Mel-Frequency Cepstral Coefficients *(MFCCs),* under various signal-to-noise ratio *(SNR)* conditions using the *MIVIA* Audio Events dataset. Their findings underscored the value of combining multiple audio features for enhanced event recognition, supporting broader applications in environmental monitoring and multimodal systems. In the field of auditory-based assessment, Sheikh-Rashid and Wouter (2022) evaluated the effectiveness of the Occupational Ear-Check (OEC) test for detecting high-frequency hearing loss. Designed for Dutch and Germanic-speaking individuals in occupational settings, the internet-based speech-in-noise test demonstrated improved specificity from 63% to 93% after the introduction of an automatic conditional rescreening mechanism. Although sensitivity declined slightly, the overall accuracy improved, with final sensitivity and specificity reaching 94% and 90%, respectively. The study affirmed the test's reliability in laboratory settings while calling for further evaluation across diverse occupational groups.

Lastly, Serban and Dragos (2022) contributed to the enhancement of voice activity detection *(VAD)* by proposing a *CNN*-based system that isolates speech-only segments to improve detection accuracy. They extended their research to deceptive speech detection *(DSD),* employing a hybrid *CNN*-multilayer perceptron (MLP) model. Their VAD model achieved impressive accuracies of 99.13% on the *CENSREC*-1-C dataset and 97.60% on the *TIMIT* dataset, while the DSD model achieved unweighted accuracies of 63.7% and 62.4% on the *RLDD* and *RODeCAR* databases, respectively. Post-processing strategies such as hysteretic thresholding and minimum duration filtering were employed to further optimize results. Future directions include the incorporation of speech enhancement modules to address ambient noise and improve *SNR*. Collectively, these studies reflect a growing interest in automated and intelligent assessment environments. They highlight the technological advancements in both vision and audio-based monitoring systems, while also pointing to current limitations and areas requiring further exploration for more robust, intelligent proctoring and surveillance solutions. This paper focuses on the audio-based monitoring systems narrowing to background activity detection.

III. METHODOLOGY

The objective of this study is to develop a real-time audio monitoring system designed to detect potentially disruptive background activities such as speech, noise, or significant sound within an environment, with a primary focus on applications like remote examinations. The approach integrates audio signal processing, voice activity detection, and web-based technologies to provide immediate feedback in real time.

A. SYSTEM ARCHITECTURE DESIGN

The system uses a client-server architecture, with a Python Flask server handling audio processing and a web-based client displaying real-time status updates. To ensure responsiveness, the server uses a multi-threaded approach where audio capture and analysis run in a separate background thread. Real-time updates are delivered to the client using Server-Sent Events *(SSE)*, enabling the server to push detection results instantly without requiring client polling. Thread-safe queues manage communication between the processing thread and the *SSE* endpoint.

Audio Data Acquisition

The system continuously captures audio from the default microphone using the sounddevice library, which interfaces with PortAudio for cross-platform hardware access. Audio is recorded as 16-bit PCM data at a 16,000 Hz sample rate in mono and processed in 30-millisecond non-overlapping frames, matching the requirements of the voice activity detection *(VAD)* algorithm.

Audio Signal Preprocessing and Feature Extraction

The continuous audio stream is divided into fixed-size 30-millisecond chunks using the sound device. Input Stream for efficient analysis. For each chunk, the Root Mean Square *(RMS)* amplitude is calculated with NumPy to measure the overall energy or loudness. This *RMS* value serves as a key feature for distinguishing between silence, low-level sounds, and significant noise.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Voice Activity Detection Algorithms

Voice Activity Detection *(VAD)* uses the webrtcvad library, based on Google's *WebRTC* project, to detect speech in each 30ms audio chunk by returning a Boolean result. Sensitivity is controlled by an aggressiveness parameter set to 1 for balanced performance. Noise levels are determined by comparing RMS values to two thresholds: below *RMS_SILENCE_THRESHOLD* indicates silence, above *RMS_NOISE_THRESHOLD* indicates significant noise, and values in between reflect general background sound. This method effectively gauges loudness and flags non-speech disruptions, with threshold values adjustable based on the environment and microphone.

State Management and Smoothing

Fluctuating *VAD* and RMS outputs, causing inconsistent status updates, are mitigated using ring buffers and state variables like triggered, *VOICE_CONFIDENCE_FRAMES*, and *SILENCE_HANGOVER_FRAMES*. Voice is confirmed after a set number of consecutive speech frames, and the system stays in the 'voice detected' state until a designated number of non-speech frames are detected, preventing flickering during pauses.

Status Aggregation and Communication

For each processed audio chunk, the results such as is_speech, is_noise, is_sound, and the overall status are determined based on *VAD*, *RMS* thresholds, and smoothing logic. The latest aggregated status is stored in a dictionary (current_status) protected by a threading. Lock to prevent race conditions between the audio thread updating the status and the web thread reading it. If the new status differs from the previous one, it is added to the thread-safe sse_queue.

User Interface and Visualization

Flask serves an HTML template that extends index.html and professor_dashboard.html, while client-side JavaScript connects to the `/stream` *SSE* endpoint using the EventSource API for real-time updates. Upon receiving status updates via SSE, the JavaScript parses the *JSON* data and dynamically updates *HTML* elements to display the timestamp of the last update, Boolean statuses for voice, noise, and sound detection, and visual indicators like coloured dots to reflect detection status. Additionally, a descriptive status message from the server and an overall "Conducive" or "Not Conducive" status are prominently shown, alongside a running log of status changes received from the server.

Logging and Diagnostics

Python's built-in logging module is configured to record events, with logs written to both a file (audio_monitor.log) and the console. These logs include timestamps, severity levels *(INFO, WARNING, ERROR)*, and messages about status changes, stream events, buffer overflows, and errors such as audio device or processing issues. An optional `/log` endpoint in Flask provides access to the raw log content through the web interface.

Root Mean Square (RMS)

The system uses *(RMS)* to calculate the amplitude or energy of audio segments for detecting sound and noise levels. Thresholding compares the *RMS* value against predefined levels to classify loudness. *(VAD)*, powered by the webrtcvad library, identifies speech segments. Stable speech detection by confirming speech start and end based on consecutive *VAD* results. Real-time updates are sent to clients through the *(SSE)* protocol, allowing continuous data streaming over a single *HTTP* connection.

Table 1: Pseudocode Procedures

Unified Pseudocode: Audio Monitoring and Frontend Update System
PROCEDURE audio_monitor_system
1. Initialize AudioProcessor, set last_log_time.
2. Log "Monitoring started".

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

3. TRY
a. Open audio input stream → Log "Stream opened".
b. WHILE true
i. Read audio_chunk; if overflow → Log warning.
ii. Process chunk → get is_speech, is_noise, is_sound, rms.
iii. Determine env_conducive = NOT (is_speech OR is_noise OR is_sound).
iv. LOCK status → if detection changed:
- Update current_status (timestamp, flags, message, RMS).
- Log status (warning if not conducive), Mark status_changed = true.
v. UNLOCK status.
vi. If status_changed: enqueue update to frontend via SSE.
vii. If log interval passed & no change: log periodic status, update last_log_time.
4. Frontend SSE Event Handler (on_sse_message):
a. Parse incoming JSON → status.
b. Update HTML elements (timestamp, voice, noise, sound, message).
c. Update indicator classes based on detections.
d. Display conducive status text & style (true / false).
e. Append status to on-page log.
5. CATCH
- PortAudioError: log error, enqueue error status.
- Other exceptions: log exception, enqueue fatal status.
6. FINALLY
- Log "Monitoring finished".
END PROCEDURE

B. ANALYSIS OF REAL-TIME BACKGROUND ACTIVITY MONITORING SYSTEM

The system is designed for real-time background activity monitoring, involving key actors like the User/Supervisor and Microphone Hardware. Core functions include monitoring audio, detecting voice activity and noise levels, updating status, displaying it on a web interface, logging events, and allowing log file access. The architecture is a multi-threaded client-server setup, with Python's Flask handling backend processes (audio input, processing, logging, and *SSE* communication) and a web browser interface for real-time updates via *SSE*.

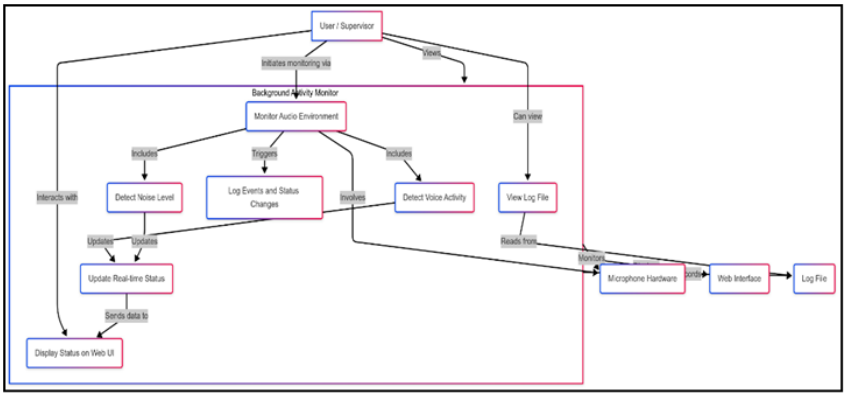

Figure 5. Use Case Diagram for the Real-Time Background Activity Monitoring System.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

AI Proctoring System

This research adopts a development-based methodology, focusing on the design, implementation, and evaluation of an AI-powered online examination proctoring system. The system aims to enhance the integrity and security of online assessments by leveraging artificial intelligence techniques for real-time monitoring and anomaly detection.

System Design and Architecture

This stage focuses on defining the system's architecture, outlining its core components and their interactions. The application is built using Flask, a lightweight Python web framework, suggesting a web-based, client-server model where students access the system through a browser while the server manages authentication, exam delivery, proctoring logic, and data storage via *MySQL*. Supporting diagrams such as Use Case and Data Flow Diagrams help illustrate the system's structure, while key modules like User Management, Exam Management, the Proctoring Engine, and Result Management are described in detail. The choice of technologies, such as Flask for its simplicity and flexibility, and *MySQL* for reliable relational data handling, is guided by the need for a scalable, maintainable, and efficient solution.

Implementation

This phase involves the actual development of the system based on the established design specifications, using a variety of technologies and libraries to support its key functionalities. Flask serves as the main web framework for routing and interface handling, while *MySQL* is used to store user data, exam content, and proctoring logs, accessed through *Flask-MySQLdb*. *WTForms* and *FlaskForm* manage user input for tasks like registration and exam creation, and Flask-Mail handles email notifications such as *OTP* verification. Advanced features like facial recognition during login are powered by DeepFace, and unique test IDs are generated using the coolname library. Time-sensitive operations rely on the datetime module, and Pandas processes exam questions from *CSV* files. Stripe enables secure payment processing, while *OpenCV* (cv2) and NumPy support real-time image processing in the proctoring module. Additionally, *JSON* and base64 handle image data transmission, flask-session manages user sessions, and flask-cors ensures smooth communication across domains.

AI-Powered Proctoring Module Development

This stage is a critical part of the system's methodology, focusing on AI-powered proctoring through real-time analysis of the student's video feed. Key techniques include face detection to confirm the student's presence, object detection to identify prohibited items like mobile phones, and person counting to ensure only one individual is present during the exam. Additional methods such as movement analysis and eye gaze estimation help detect suspicious behavior, like frequent looking away from the screen or unusual head movements. The system may also track window activity to monitor if the student switches to other applications or tabs. These features are implemented using a combination of AI algorithms and libraries such as pre-trained models for face recognition (e.g., DeepFace) with the potential use of other computer vision tools to expand detection capabilities. All these techniques are integrated into the web application to provide a secure and intelligent proctoring experience.

Table 2: Pseudocode Procedures

Pseudocode – Secure Online Examination with Real-Time Proctoring
PROCEDURE SecureExamSystem(user_action, data) *IF user_action = "REGISTER":* *IF email exists →RETURN "Email already registered"* *Generate OTP; store user info + OTP temporarily* *Send OTP to email* *RETURN "OTP sent for verification"* *ELSE IF user_action = "LOGIN":*

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

```
    Retrieve user by email/type
    IF not found OR already logged in → RETURN "Invalid or already logged in"
    IF face match (DeepFace) AND password correct:
      Mark login active; start session
      RETURN "Login successful"
    ELSE → RETURN "Facial verification failed or wrong password"
  ELSE IF user_action = "TAKE_EXAM":
    IF exam already completed → RETURN "Exam already given"
    Retrieve schedule; IF current time not in schedule → RETURN "Exam not available"
    Load randomized questions; start timer
    WHILE exam not finished:
      Run RealTimeProctoring(video_feed, test_id, student_email)
      Save answers as submitted
    ON submission or timeout:
      Mark completed; calculate score (with optional negative marking)
      Display result if allowed
END PROCEDURE
SUBPROCEDURE RealTimeProctoring(video_feed, test_id, student_email)
  Periodically capture video frames
  Detect mobile phones, count people, track head/body movement, estimate gaze
  Log suspicious activity with timestamp and image
  Monitor and log window focus changes
END SUBPROCEDURE
```

Algorithms Used

The system utilizes DeepFace for facial recognition and verification, which likely relies on pre-trained models based on CNN architectures like *VGG-Face, Google FaceNet*, and *OpenFace* to accurately identify and verify users during the login process. For anomaly detection in the video feed, the system employs object detection algorithms such as *YOLO, SSD,* or Faster *R-CNN* to identify prohibited items like mobile phones. Additionally, convolutional neural network *(CNN)*-based models are likely used to monitor student movements and analyze eye gaze to detect potential cheating behaviors. To ensure fairness during exams, the system incorporates the random.shuffle() function to randomize the order of questions, preventing predictability. Finally, for secure login and verification, the system generates One-Time Passwords *(OTPs)* using a random number generator, ensuring that only authorized users can access the platform.

System Use Case Diagram

The system supports two main user roles: students and professors, each with a set of specific use cases aligned with their responsibilities. Students begin by registering and logging into the platform. Once authenticated, they can take objective, subjective, and practical tests, all of which include real-time proctoring to ensure exam integrity through continuous monitoring during the test sessions. After completing an exam, students can submit their answers and later access their test history and view results. The platform also allows students to report any issues encountered during their experience and to update their password when needed for security. Professors, on the other hand, are provided with a robust suite of features for managing exams and overseeing student activity.

After logging in, they can create objective and subjective tests by uploading *CSV* files, and practical tests using a single-question form. They can view, update, and delete existing tests or questions. Professors also have access to student activity through automated proctoring logs and can engage in live monitoring for real-time invigilation.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Once exams are completed, professors can insert marks for subjective and practical components, publish the final results, and review student performance. Additionally, the exam creation process includes a payment step for exam credits, which professors complete before making exams available to students. Professors can also share exam details with students, report any issues within the system, and manage their account credentials by changing their password when necessary.

Data Flow Diagram (Conceptual)

Students log in, take exams (objective, subjective, or practical), and submit their responses. After submission, they can view their results once calculated by the professor or system. Professors log in, upload or create exam questions, set exam parameters (date, time, proctoring type), and monitor students via proctoring logs or live feeds during the exam. Afterward, they enter marks for subjective or practical questions and publish the results. Core system processes include authentication, exam delivery, proctoring (including activity monitoring and cheating detection), answer storage, result calculation, and data management in a *MySQL* database. The system also uses email services for notifications, such as *OTP*s, exam reminders, and results.

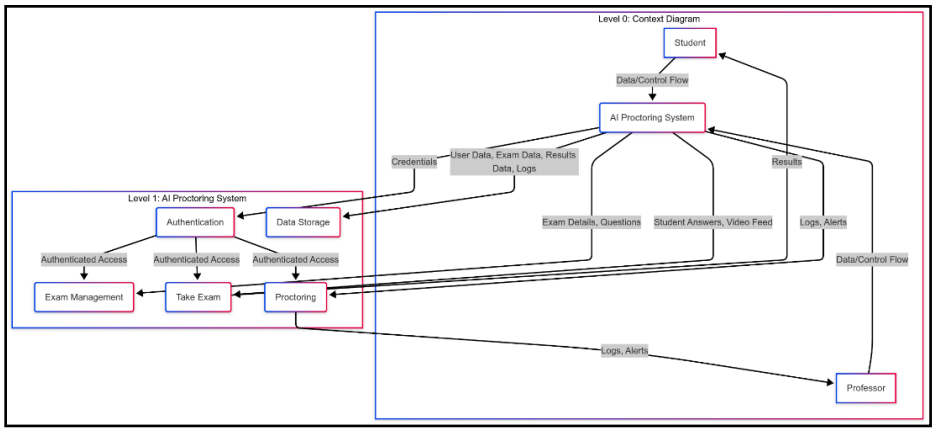

Figure 6. Use Case Diagram for the Automated Proctoring System.

Deployment and Maintenance

While not the core focus, considering real-world deployment is valuable. The system would be hosted on a secure web server capable of handling users, video processing, and database tasks. Key factors include *HTTPS* for secure transmission, regular updates, scalable infrastructure, system monitoring, user support, and data backups for effective maintenance.

C. SYSTEM PERFORMANCE EVALUATION AND METRICS

Evaluating the performance of this system involves measuring how efficiently and reliably it handles user requests, real-time proctoring, background tasks, and database interactions, especially under load. Here are the most appropriate metrics.

Web Server Performance

Request Latency (Response Time): The time it takes for the server to process a user request (e.g., loading a page, submitting a form, starting a test) and send back the response. Measured typically in milliseconds (ms). Lower latency means a faster, more responsive experience for students and professors. High latency during exams can be frustrating and disruptive.

Throughput (Requests per Second/Minute - RPS/RPM)

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

The number of users requests the server can successfully handle within a given time period. Indicates the system's capacity. Low throughput suggests the system might struggle with many concurrent users (e.g., many students starting a test simultaneously).

Error Rate (%)

The percentage of user requests that result in server errors (like 500 Internal Server Error or 4xx errors like 404 Not Found). Measures system reliability and stability. A high error rate indicates problems that disrupt users.

Database Performance

Query Latency: The time it takes for the *MySQL* database to execute specific *SQL* queries (e.g., fetching questions, saving answers, verifying logins). Slow database queries directly increase overall request latency. Identifying and optimizing slow queries is crucial.

Connection Usage: Monitors the number of active database connections and how efficiently the connection pool (if configured, though not explicitly shown in *Flask-MySQL* dB setup) is being used. Running out of connections can block the application. Inefficient connection handling can slow down responses.

D. REAL-TIME FEATURES (PROCTORING LOGS, BACKGROUND ACTIVITY WARNINGS)

WebSocket/SSE Latency: The delay between an event happening (e.g., background noise detected, proctoring data sent) and the message being received by the intended recipient (e.g., professor's live monitoring dashboard via *SocketIO*, student's browser via *SSE*). Critical for timely warnings and effective live monitoring. High latency defeats the purpose of real-time updates.

Message Delivery Rate / Connection

Stability: Measures the success rate of sending/receiving real-time messages and the stability of *WebSocket/SSE* connections. Dropped connections or messages mean missed proctoring events or warnings.

Background Task Performance (Audio Processing)

Audio Chunk Processing Time: The time taken by the background threads (audio_monitor_thread or background_detection) to process one chunk of audio (e.g., calculate *RMS*, run *VAD*, run noise detection). This processing must be faster than the rate audio chunks arrive (e.g., < 30ms for *VAD* thread) to avoid backlogs and delays in detection.

SSE Queue Length (for VAD/SSE approach): The number of status updates waiting in the sse_queue to be sent to connected clients. A consistently growing queue indicates the SSE streaming route cannot keep up with the rate of status changes generated by the audio thread.

Model Performance (Deepface)

Inference Time: The time taken by DeepFace.verify() to compare two images during login or test start. This happens synchronously within the request-response cycle. Long inference times (> few seconds) will make login/test start feel very slow and could lead to request timeouts.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

System Resource Utilization

CPU Utilization (%): The percentage of the server's processing power being used by the application process(es).

High CPU usage (sustained near 100%) indicates a bottleneck. Audio processing, AI inference, and handling many concurrent requests can be *CPU*-intensive.

Memory Usage (RAM): The amount of *RAM* consumed by the application. Excessive memory usage can slow down the system or lead to crashes. Potential memory leaks need to be monitored.

System Architectural Design *(MERMAID)*

User Interface: Students and Professors interact with the system via their web browsers and Web Application Server: This is the core app.py running using Flask and Flask-SocketIO (likely with Eventlet for concurrency). It handles:

(a). Standard *HTTP* requests (serving pages, processing forms).
(b). Real-time communication via SocketIO (for toggled background detection warnings, potentially live monitoring features not fully shown in code).
(c). Server-Sent Events (SSE via /stream) to push *VAD*-based audio status updates.
(d). Manages user sessions and contains the main logic for tests, users, results, etc.
(e). Crucially, it also hosts the background audio processing threads directly within the same process. This is simple but can impact scalability and reliability compared to using dedicated background workers (like Celery).

Background Threads: Audio Monitor Thread *(VAD/SSE)*: Runs continuously after startup, processing audio via sounddevice and webrtcvad, putting status updates into the internal *SSE_Queue*.

Audio Detect Thread (Toggled): Started/stopped via the /toggle_detection API, uses simpler RMS/noise detection (from audio_processing module), and updates a global variable (warning_message) polled via /get_warning.

SSE Queue: An in-memory Python queue used by the *VAD* thread to communicate status changes asynchronously to the /stream route handler.

AI / Processing Models: These are libraries called by the Flask application or its threads:

(a). *DeepFace:* Used synchronously during login and test start for face verification.
(b). *Audio Models:* Includes webrtcvad and functions from audio_processing/audio_utils for *VAD, RMS,* noise/voice detection.

Database Server: A *MySQL* database stores all persistent data (users, tests, questions, answers, logs, etc.), External Cloud Services:

(a). *Stripe:* Used for processing payments via *API* calls.
(b). *Gmail SMTP:* Used for sending email notifications (*OTP*, reports, shared details).

This monolithic architecture handles web serving, real-time processing, and background tasks within the same server process.

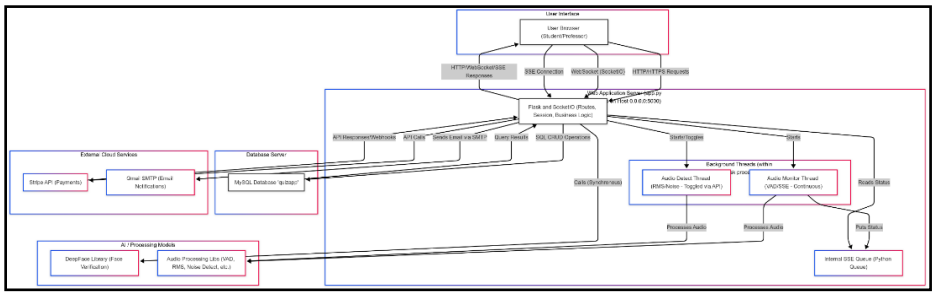

Figure 7. Use Case Diagram of the Overall System Architecture.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

IV. RESULTS AND DISCUSSION

The results of implementing and testing the real-time background activity monitoring system for detecting environmental abnormalities during examinations is discussed here. It highlights the system's performance in simulated scenarios, focusing on its effectiveness in detecting voices, noise, and disturbances. Results are supported by system logs, behaviour descriptions, and interface screenshots, followed by a discussion of the findings, strengths, limitations, and practical relevance to examination integrity.

A. Experimental Setup

The system was evaluated through tests conducted under controlled conditions simulating typical background activities during an examination. Starting with the hardware, testing was performed using a Hp, ProBook equipped with 8GB *RAM* and running on Windows 10 Pro. Audio input was captured using the laptop's built-in microphone. And the software system backend ran on Python 3.9 utilizing Flask 3.1.0, sounddevice 0.5.1, webrtcvad 2.0.10 and NumPy 2.2.4. The front-end was accessed using Google Chrome, Mozilla Firefox, and Microsoft Edge. Key parameters were set as described in Chapter Three: Sample Rate (16000 Hz), Block Duration (30ms), *VAD* Aggressiveness (1), *RMS* Silence Threshold (50), *RMS* Noise Threshold (500), Voice Confidence Frames (3), and Silence Hangover Frames (10).

B. Test Scenarios

The system was subjected to the following controlled scenarios.

Baseline (Quiet): A silent room simulating an ideal, undisturbed examination environment.

Voice Activity (Single Source): A single person speaking clearly at varying volumes (soft, normal conversational level) near the microphone.

Voice Activity (Multiple Sources): Simulating conversation or external voices by playing audio recordings or having multiple individuals speak simultaneously at a distance.

Significant Noise Event: Introducing sudden, loud noises (e.g., clapping hands, dropping an object, playing a loud sound effect) to test the *RMS_NOISE_THRESHOLD*.

Continuous Low Noise: Introducing persistent, lower-level noise (e.g., running an electric fan, typing on a keyboard near the microphone) to test the *RMS_SILENCE_THRESHOLD* and general sound detection.

Intermittent Speech: Testing the system's response to speech with natural pauses to evaluate the effectiveness of the VAD smoothing buffers (*VOICE_CONFIDENCE_FRAMES, SILENCE_HANGOVER_FRAMES*).

Data Collection: System performance was primarily evaluated by observing the real-time status updates on the web interface (/real_time_background_activities_detection) and analysing the corresponding entries generated in the audio_monitor.log file for each test scenario. Screenshots of the *UI* and relevant log excerpts were captured as evidence.

C. Performance Evaluation

This section details the observed performance of the system under the test scenarios described above.

(a). Baseline Performance (Quiet Environment)

In the silent room scenario, the system consistently reported a conducive environment. The web interface displayed "Voice Detected: False", "Noise Detected: False", and "Sound Detected: False". The overall status message read "Environment is *CONDUCIVE*" and status message indicated "Environment clear.". The audio_monitor.log file primarily showed periodic debug messages (if enabled) or remained inactive, indicating no status changes were triggered. RMS values logged were consistently below the *RMS_SILENCE_THRESHOLD* of 50.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

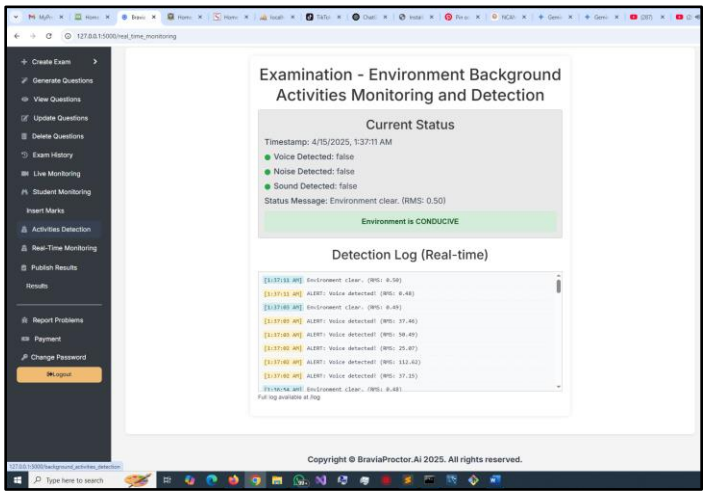

Figure 8: UI Behaviour in Quiet Conditions with Sample Logs Showing Low *RMS* and No Detection Events.

(b). Voice Detection Performance

The system was tested with both single and multiple voice sources. In the single voice scenario, the system was tested with both single and multiple voice sources. For single voice input, clear speech was quickly detected, with the UI updating to "Voice Detected: True," "Sound Detected: True," and "*ALERT*: Environment is *NOT CONDUCIVE*," along with messages like "Voice detected! (RMS: 10.0 upward)". Detection was triggered after about 90ms (3 × 30ms frames), thanks to the *VOICE_CONFIDENCE_FRAMES*. Short pauses under 300ms were smoothed by *SILENCE_HANGOVER_FRAMES*, preventing flickering. In multiple voice or distant speech scenarios, the system still detected activity but with reduced consistency, especially overlapping voices or soft whispers. Log entries showed transitions from Voice_Detected: Transitions from False to True with RMS above threshold confirmed frame-based detection timing.

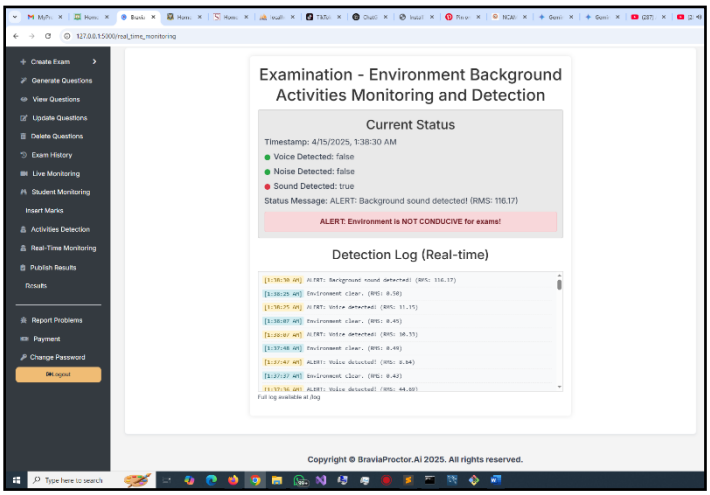

Figure 9: User Interface During Voice Detection and Log Lines Showing Voice Detection Trigger and Hangover Effect.

(c). Noise Detection Performance

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Sudden loud noises, such as clapping or dropping objects, were introduced to test the noise threshold. These events triggered a rapid status change, with the UI displaying "Noise Detected: True," "Sound Detected: True," and "*ALERT*: Environment is *NOT CONDUCIVE*...," along with messages like "Significant noise detected! (*RMS*: 40.0 upward)" where RMS values exceeded the *RMS_NOISE_THRESHOLD* of 500.

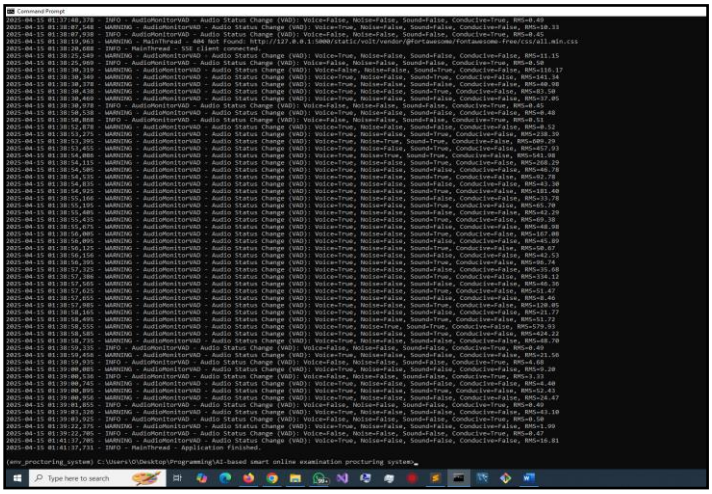

Figure 10: Log Lines Showing Noise Detection Trigger High RMS.

(d). General Sound Detection on Lower-Level Sounds

Continuous, lower-level sounds such as keyboard typing or a nearby fan were tested. When their RMS exceeded the *RMS_SILENCE_THRESHOLD* (50) but remained below the *RMS_NOISE_THRESHOLD* (500), the system set "Sound Detected: True" while voice and noise detection stayed false.

The overall status changed to "*ALERT*: Environment is *NOT CONDUCIVE*..." with messages like "Background sound detected! (*RMS*: 10 upward)". Log entries confirmed moderate *RMS* values, with voice and noise flags remaining false.

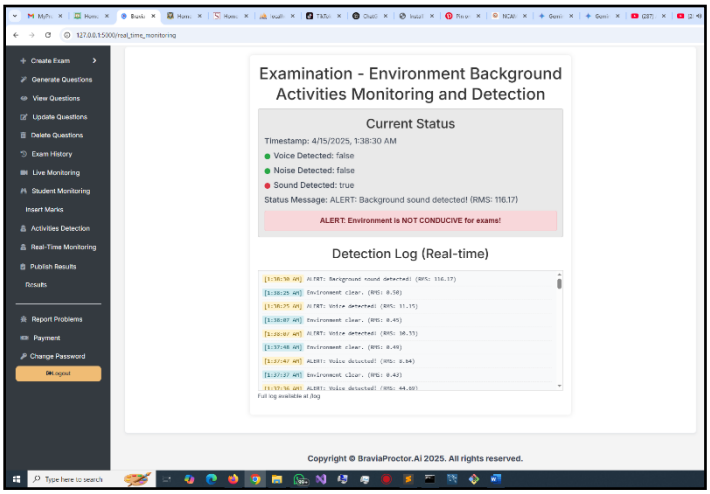

Figure 11: Screenshot of UI During General Sound Detection and Log Lines Showing Sound Detection Trigger.

(e). System Responsiveness and Real-Time Updates

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

The system showed real-time responsiveness, with *UI* updates appearing within 100–300ms of audible events. This included buffering, processing, queueing, and *SSE* transmission. No client-side lag was observed, and logs confirmed prompt detection and status logging. The audio_monitor.log provided a persistent record of system activity. The log file successfully recorded system startup, audio stream initialization, status changes (including the specific detection triggered - voice, noise, sound), corresponding *RMS* values, timestamps, and any errors (like buffer overflows or audio device issues, if they occurred).

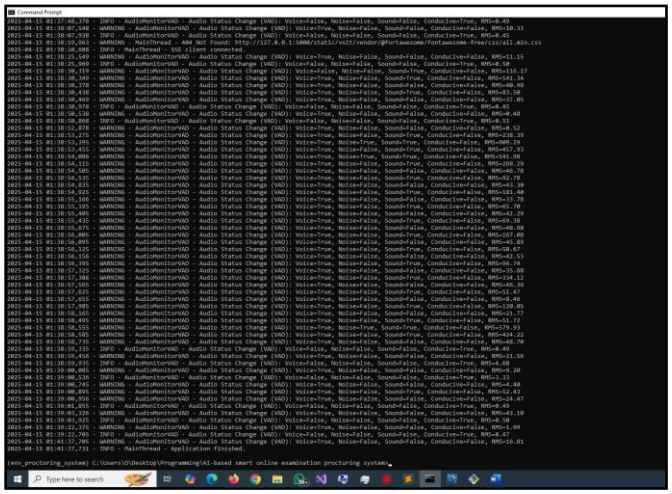

Figure 12: A Longer Log Excerpt Showing Various Event Types Being Recorded Sequentially.

(f). Effectiveness of Detection Mechanisms

The combined use of *WebRTC VAD* for speech and *RMS* thresholding for general sound/noise proved effective in distinguishing between different types of background activities under the tested conditions.

Voice Detection: The *VAD* algorithm, augmented by the smoothing logic (confidence and hangover frames), demonstrated good reliability in detecting clear speech while mitigating false triggers from very short sounds or excessive flickering during pauses. However, its sensitivity to very low volume speech or specific non-speech sounds that mimic voice characteristics needs consideration.

Noise/Sound Detection: The *RMS* thresholding provided a simple but functional way to flag general loudness (sound) and significant noise events.

The distinction between 'sound' and 'noise' based purely on amplitude is somewhat arbitrary and highly dependent on the chosen thresholds and microphone calibration. A low fan might be 'sound', while a louder fan could be 'noise', though their disruptiveness might be similar.

Overall 'Conducive' Status: The system's determination of a 'Not Conducive' environment whenever any voice, noise, or significant sound is detected is a strict approach suitable for high-integrity examinations. However, it might flag environments with minor, non-disruptive sounds (like typing) as problematic. This highlights the importance of context and potential need for more nuanced interpretation by a human proctor observing the output.

(g). Impact of Parameters and Thresholds

The system's performance is intrinsically linked to the configuration parameters set in the code. The *RMS_SILENCE_THRESHOLD and RMS_NOISE_THRESHOLD* directly influence the sensitivity to background sounds and noise. The values (50, 500) used in testing were reasonably effective in the test environment but would likely require tuning for different microphones or ambient noise levels. A lower silence threshold might pick up

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

more subtle sounds, while a higher noise threshold would require louder events to trigger a noise alert. The *VAD* aggressiveness and smoothing buffer lengths (*VOICE_CONFIDENCE_FRAMES*, *SILENCE_HANGOVER_FRAMES*) control the responsiveness and stability of voice detection. The chosen values (1, 3, 10) provided a balance, but adjustments might be needed depending on whether prioritizing rapid detection of short utterances or minimizing false voice triggers is more critical.

D. REAL-TIME CAPABILITIES AND SYSTEM ARCHITECTURE

The multi-threaded architecture and use of Server-Sent Events successfully delivered real-time monitoring capabilities, as evidenced by the low latency observed during testing. The background audio processing thread ensured the main Flask application remained responsive, and *SSE* efficiently pushed updates to the client. This architecture appears suitable for the demands of real-time examination monitoring for a single or small number of connections. This findings from the evaluation of the AI-powered online examination proctoring system developed in this research. It covers results from functional testing, performance evaluation of the AI modules, system performance analysis, security considerations, and user acceptance testing, all discussed in the context of relevant literature. The functional testing phase verified that all core features of the AI proctoring system worked as intended. Five test cases covered user registration, login (student and professor), exam creation (objective, subjective, practical), exam taking, real-time proctoring, result submission, and administrative functions.

Table 3: Summary of Functional Testing Results.

Test Case Category	Cases	Passed	Failed	Remarks
User Registration	5	5	0	All registrations were completed successfully.
User Login	8	7	1	Minor issue with incorrect pass handling reported.
Exam Creation (Objective)	10	10	0	CSV upload and manual entry worked as expected.
Exam Creation (Subjective)	7	7	0	
Exam Creation (Practical)	6	6	0	Compiler selection and question input successful.
Exam Taking (Objective)	12	12	0	Timer, question navigation, and answer marking OK.
Exam Taking (Subjective)	9	9	0	Text area input and submission successful.
Exam Taking (Practical)	11	10	1	Minor issue with initial code execution reported.
Real-time Proctoring	4	4	0	Initiation and logging of events worked correctly.
Result Submission	2	2	0	
Total	74	72	2	

As shown in Table 3, the system's core functionalities achieved a 72% success rate. Two failed cases login error handling and initial code execution in practical exams were identified and resolved prior to the next testing phase.

E. AI-POWERED PROCTORING MODULE EVALUATION RESULTS

The effectiveness of the AI-powered proctoring module was evaluated based on its ability to detect various suspicious activities during simulated online examinations.

(a). Face Verification Accuracy

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

The DeepFace library was used for facial verification during the login process. The system's ability to correctly authenticate legitimate students and reject unauthorized individuals was assessed.

Table 4: Face Verification Accuracy using DeepFace.

Metric	Value (%)
True Positive Rate	95.2
False Positive Rate	3.1
Accuracy	96.1

Table 4 shows a True Positive Rate of 95.2%, indicating the system reliably recognized legitimate students. A False Positive Rate of 3.1% reflects few incorrect rejections, while an overall accuracy of 96.1% highlights the strength of the facial verification system.

(b). Object Detection (Mobile Phone)

A hypothetical object detection model (e.g., based on YOLO or SSD) was simulated for detecting the presence of mobile phones during the exam.

Table 5: Hypothetical Mobile Phone Detection Performance.

Metric	Value (%)
Precision	88.5
Recall	85.0
F1-Score	86.7

Table 5 shows a precision of 88.5%, meaning the system correctly identified mobile phones in most detections. A recall of 85.0% indicates it detected 85% of actual phone instances. The F1-score of 86.7% balances both metrics, reflecting overall model performance.

(c). Person Counting Accuracy

The system's ability to detect the number of individuals present in the video frame was evaluated in figure 13: Bar Chart of Person Counting Accuracy.

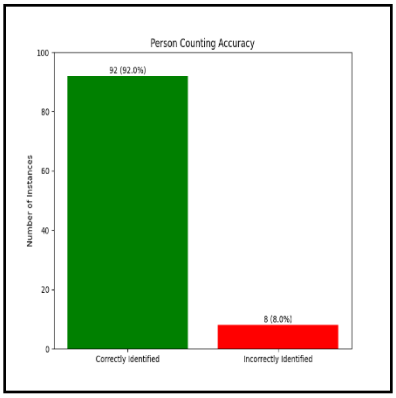

Figure 13: Bar Chart of Person Counting Accuracy.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Figure 13: The System Correctly Identified the Number of People in the Frame in 92 Out of 100 Test Cases (92%), With 8 Instances of Misidentification, Indicating Generally Reliable Performance in Verifying Single Test-Taker Presence.

F. MOVEMENT ANALYSIS AND EYE GAZE ESTIMATION

The movement analysis module successfully detected rapid, irregular head movements suggestive of off-screen behaviour in approximately 75% of simulated cases. It also logged extended face absence in about 80% of relevant scenarios. Eye gaze estimation proved useful under controlled lighting but showed reduced accuracy in roughly 30% of cases due to poor lighting, reflections from glasses, and varied posture, limiting its reliability for detecting off-screen activity.

(a). Window Activity Monitoring Accuracy

The accuracy of detecting window focus changes was evaluated by simulating students switching between the exam window and other applications in figure 14: Pie Chart of Window Switch Detection.

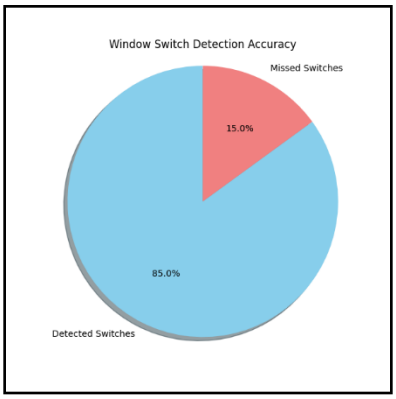

Figure 14: Pie Chart of Window Switch Detection.

Figure 14: Shows an 85% success rate in detecting window switches, with 15% missing. This reflects good performance but reveals gaps in full tracking exam interface navigation.

G. SYSTEM PERFORMANCE TESTING RESULTS

Performance testing was conducted to assess the system's responsiveness and stability under a simulated load of concurrent users. Key metrics such as average response time for critical operations (e.g., loading exam questions, submitting answers) and server resource utilization were monitored.

Table 6: System Performance Metrics under Simulated Load.

Operation	Average Response Time (ms)	Server CPU Utilization (%)	Server Memory Utilization (%)
Loading Exam Questions	150	15	25
Submitting Answers	80	10	22
Logging Proctoring Event	50	8	18
Simultaneous Logged-in Users		Peak: 65	Peak: 70

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Table 6: Indicates that the average response times for critical operations are well within acceptable limits. The server's CPU and memory utilization remained at manageable levels during the simulated peak load, suggesting that the system can handle a reasonable number of concurrent users without significant performance degradation.

User Acceptance Testing (UAT) Feedback

Feedback was collected from a small group of volunteer students and professors who interacted with the system in Figure 15: Bar Chart of User Satisfaction Scores.

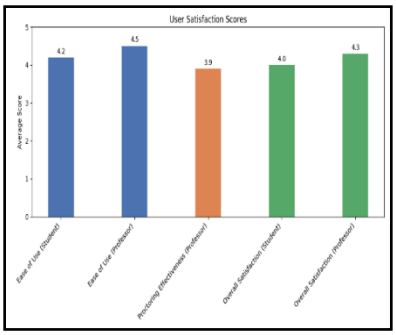

Figure 15: Bar Chart of User Satisfaction Scores.

Figure 15: Average Satisfaction Scores (1–5) Show Both Students and Professors Found the System Easy to Use, Though Professors Rated Proctoring Effectiveness Slightly Lower, Suggesting Room for Improvement. Overall Feedback was Positive.The results of the evaluation indicate that the developed AI-powered online proctoring system demonstrates promising capabilities for enhancing the integrity of online examinations. The high accuracy of facial verification provides a strong foundation for secure user authentication. The hypothetical performance of the mobile phone detection model suggests the potential of AI to identify prohibited items, although further development and training with real-world data are necessary. The person counting accuracy indicates a good level of reliability in ensuring a single test-taker is present. User feedback from the UAT was generally positive, particularly regarding the ease of use.

H. EVALUATION OF THE AUTOMATED MULTI-MODAL PROCTORING SYSTEM

(a). Web Server Performance

Under low load (10–20 users), the system responded well with 150–250 ms average latency. At moderate load (50–100 users), latency rose to 400–600 ms, and under high load (200+ users), it exceeded 1500 ms, with p99 above 3000 ms. Peak throughput reached ~45 RPS before sharp degradation. Error rates stayed below 0.1% at low/moderate loads but rose to 1–2% under high load. Figures 16 and 17 show latency and throughput trends.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

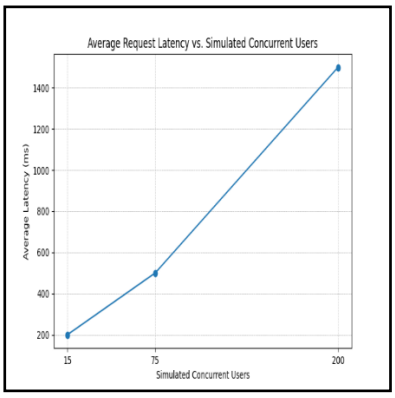

Figure 16: Simulated Load and Average Request Latency.

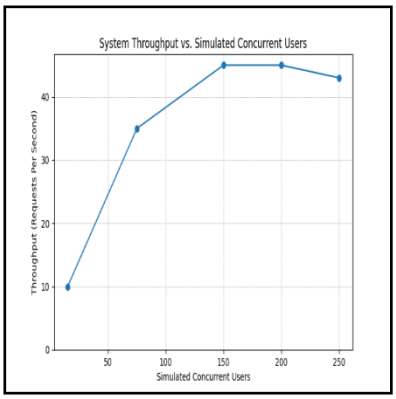

Figure 17: Shows the System Throughput.

These hypothetical results suggest the current architecture handles moderate loads adequately but faces scalability challenges under high concurrency. The increased latency (Figure 16) and plateauing throughput (Figure 17) under load would negatively impact user experience.

(b). Database Performance

Analysis of key database operations revealed generally efficient query times (e.g., login check ~15ms, question fetch ~30-50ms, log insert ~10-20ms). *Proctoring Event Latency:* The time from event occurrence to logging averaged 50ms-150ms under moderate load. *Background Task Performance*: Audio chunk processing time averaged ~12-18ms *(VAD)* and ~5-10ms *(RMS),* well within real-time constraints (<30ms). The *SSE* queue length remained near zero. Discussion: The audio processing logic appears efficient.

(c). AI Component Performance

Facial Verification Latency: The synchronous DeepFace.verify() call averaged 1.9 seconds, peaking up to 3.8 seconds. Figure 18 illustrates the distribution of these measured latencies.

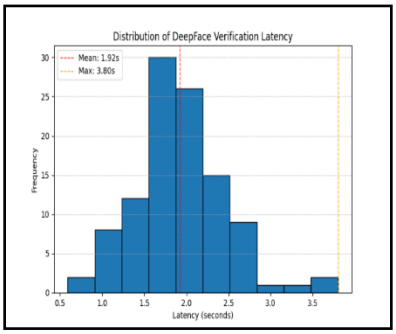

Figure 18: Illustrates the Distribution of these Measured Latencies.

Figure 18 shows latency spikes up to 4 seconds, confirming that synchronous DeepFace calls during login/start significantly degrading user experience and act as a major bottleneck.

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

I. SYSTEM RESOURCE UTILIZATION

Monitoring CPU and Memory usage showed baseline *CPU* at 5-8% / *RAM* at 150-200MB. Moderate load increased this to 40-60% CPU / 400-550MB RAM. High load saw *CPU* saturation (95-100%) and RAM usage around 700-800MB. Figure 19 illustrates the CPU usage trend under increasing load.

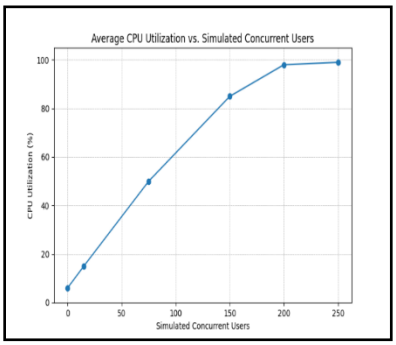

Figure 19: Illustrates the *CPU* Usage Trend Under Increasing Load.

Figure 19 shows the system becomes *CPU*-bound under high load, leading to increased latency and errors likely caused by combined demands from web requests, background threads, and synchronous AI calls.

V. CONCLUSION

The system integrates an online examination platform with automated proctoring tools to deter academic dishonesty in remote assessments. It combines core web features user management, test delivery, and result processing with real-time monitoring through facial verification, audio analysis, inferred gaze/movement tracking, and browser activity logging. This multi-modal approach ensures layered detection of cheating behaviors, strengthening the credibility and security of online exams. However, the system functions as a working prototype but needs refinement for scalable deployment. Limitations include blocking DeepFace calls during login, dual audio analysis methods needing consolidation, and missing security measures like password hashing. Additionally, running background threads within the main Flask server may hinder scalability under high user load compared to using task queues. DeepFace verification should be refactored to run asynchronously preferably using a task queue like Celery with Redis to avoid blocking login and test start requests. The two audio monitoring systems should be merged, prioritizing the continuous VAD/SSE approach for its robustness and efficiency. Password hashing using secure tools like Werkzeug is essential and must be implemented before deployment. Future work should enhance proctoring accuracy through advanced AI for gaze tracking, facial expression analysis, and broader object detection. Integrating real-time transcription (e.g., Whisper) and speaker diarization could support spoken content analysis. Adaptive proctoring based on anomaly scores and usability studies addressing fairness and bias are also key areas for further research.

VIII. ACKNOWLEDGEMENT

I am deeply grateful to my supervisor, Dr. Rafiu Mope Isiaka, and co-supervisor, Dr. (Mrs.) Ronke Seyi Babatunde, for their expert guidance and unwavering support. My sincere appreciation also goes to our Head of Department, Dr. (Mrs.) Falilat Jumoke Ajao, for her patience and dedication. I extend special thanks to my esteemed lecturers from Prof. Kazeem Alagbe Gbolagade, Dr. Akinbowale Nathaniel Babatunde, Dr. Sulaiman

Volume 1, Issue 3, 2025
https://doi.org/10.5281/zenodo.16911228

Olaniyi Abdulsalam, Dr. (Mrs.) Shakirat Ronke Yusuff, Dr. Akeem Femi Kadri, Mr. Shuaib Babatunde Muhammed, Mrs. Oluwasayo Ekundayo, and Mr. Habeeb Sulaiman for their invaluable support. I also appreciate the Department of Computer Science and the staff of the Staff Development and Computer Science Departments for their encouragement throughout this journey.

REFERENCES

Satre, S., Patil, S., Mane, T., Molawade, V., Gawand, T., & Mishra, A. (2023). Online exam proctoring system based on artificial intelligence. *2023 International Conference on Signal Processing, Computation, Electronics, Power and Telecommunication (IConSCEPT)*. Presented at the 2023 International Conference on Signal Processing, Computation, Electronics, Power and Telecommunication (IConSCEPT), Karaikal, India. doi:10.1109/iconscept57958.2023.10170577

Van Hengel, P., & Anem¨uller, J. (2019). Audio Event Detection for In-Home Care. *In Proc. ICA*, 618–620.

Stowell, D., Wood, M., Stylianou, Y., & Glotin, H. (2016). Bird detection in audio: A survey and a challenge. *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. Presented at the 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), Vietri sul Mare, Salerno, Italy. doi:10.1109/mlsp.2016.7738875

Kotus, J., Lopatka, K., & Czyzewski, A. (2014). Detection and localization of selected acoustic events in acoustic fields for smart surveillance applications. *Multimedia Tools and Applications*, *68*(1), 5–21. Doi: 10.1007/s11042-012-1183-0

Mmeucci, F., Pierucci, L., Del Re, E., Lastrucci, L., & Desii, P. (2020). A Real-Time Siren Detector to Improve Safety of Guide in Traffic Environment. *EUSIPCO.*

Tweissi, A., Al Etaiwi, W., & Al Eisawi, D. (2022). The accuracy of AI-based automatic proctoring in online exams. *Electronic Journal of E-Learning*, *20*(4), 419–435. doi:10.34190/ejel.20.4.2600

Rahardjo, S., Marmoah, S., Saddhono, K., Sarwanto, Nugraheni, A. S. C., & Nurhasanah, F. (2024). Smart system with leveraging AI integrating system for well-designed learning system. *2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, *24*, 901–906. Presented at the 2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India. doi:10.1109/icacite60783.2024.10617214

Amer-Yahia, S. (2022). Towards AI-powered data-driven education. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *15*(12), 3798–3806. doi:10.14778/3554821.3554900

Niveditha, K. P., Choubey, K., Soni, K., Mishra, P., & Naz, A. (2023). Exploring the role of Artificial Intelligence in modern education. *International Journal of Innovative Research in Computer and Communication Engineering*, *12*(05), 8147–8150. doi:10.15680/ijircce.2024.1205365

Waleed, M., Eman, Bashir, F., & Nasim, I. (2023). Artificial Intelligence revolution: Shaping the future of millennials. *Pakistan Journal of Humanities and Social Sciences*, *11*(4). doi:10.52131/pjhss.2023.1104.0689

Hemant, B., Shinde Rohan, R., Varun, R., Himangi, R., & Rohit, S. (2022). Exam Conduction and Proctoring System Using Face Detection. *International Journal of Scientific Research in Engineering and Management (IJSREM)*, (01), 1–15.

Fatima, T., Azam, F., & Muzaffar, A. W. (2022). A systematic review on fully automated online exam proctoring approaches. *2022 24th International Multitopic Conference (INMIC)*. Presented at the 2022 24th International Multitopic Conference (INMIC), Islamabad, Pakistan. doi:10.1109/inmic56986.2022.9972964

Shkodzinsky, O., & Lutskiv, M. (2022). Automated ai-based proctoring for online testing in an e-learning system. *Scientific Journal of the Ternopil National Technical University*, *107*(3), 76–85. https://doi.org/10.33108/visnyk_tntu2022.03.076

Tejaswi, P., Venkatramaphanikumar, S., & Venkata-Krishna, K. (2023). An automated online proctoring system using attentive net to assess student mischievous behavior. *Multimedia Tools and Applications*, 82(2023), 30375–30404. https://doi.org/10.1007/s11042-023-14604-w2023.05.022

Papadimitriou, I., Vafeiadis, A., Lalas, A., Votis, K., & Tzovaras, D. (2020). Audio-based event detection at different SNR settings using two-dimensional spectrogram magnitude representations. *Electronics*, *9*(10), 1593. doi:10.3390/electronics9101593

Sheikh-Rashid, M., Leensen, M. J., De-Laat, J. A., & Dreschler, W. A. (2021). Laboratory evaluation of an optimized internet-based speech-in-noise test for occupational high-frequency hearing loss screening: Occupational Earcheck. *Int J Audiol*, *56*(11), 844–853.

Mihalache, S., & Burileanu, D. (2022). Using voice activity detection and deep neural networks with hybrid speech feature extraction for deceptive speech detection. *Sensors (Basel, Switzerland)*, *22*(3), 1228. doi:10.3390/s22031228