AI-Based Online Exam Proctoring System: A Real-Time Cheating Detection Framework using Computer Vision, Deep Learning, and Browser-Based Behavioral Analysis

R. Keerthana, S. Padmaja, K. Kavibharathi
Department of Computer Science and Engineering
E.G.S. Pillay Engineering College, Nagapattinam, Tamil Nadu, India
keerthanaragupathy9@email.com, padmajasingaravelu@email.com,
kavibharathikarthikeyanb@email.com

How to Cite this Article:

Keerthana, R., Padmaja, S. & Kavibharathi, K. (2026). AI-Based Online Exam Proctoring System: A Real-Time Cheating Detection Framework using Computer Vision, Deep Learning, and Browser-Based Behavioral Analysis. International Journal of Creative and Open Research in Engineering and Management, 02(04). https://doi.org/10.55041/ijcope.v2i4.1045

https://doi.org/10.55041/ijcope.v2i4.1045

Abstract

The rapid expansion of online education following the COVID-19 pandemic has elevated the demand for robust, automated mechanisms to safeguard academic integrity. Traditional invigilation methods fail to scale in distributed learning environments, while existing commercial proctoring platforms suffer from high costs, opaque proprietary architectures, algorithmic bias, and significant privacy concerns. This paper presents an AI-based online examination proctoring system that integrates computer vision, deep learning-based object detection, and browser-based behavioral analysis to detect cheating behaviors in real time. The proposed framework employs MediaPipe for face detection and facial landmark extraction, YOLOv8 Nano for detecting prohibited objects such as mobile phones, and a Perspective-n-Point (PnP) method for three-dimensional head-pose estimation. Browser tab-switching behavior is monitored using the W3C Page Visibility API. Generated alerts are persisted in a MongoDB database and propagated to an administrative dashboard through WebSocket-based Socket.IO communication with sub-500 ms latency. Experimental evaluation across 47 participants and six simulated examination sessions yielded a weighted precision of 93.7%, recall of 91.2%, and F1-score of 92.4%. The system provides a scalable, transparent, and cost-effective open-source alternative for secure online examinations.

Keywords—*Online proctoring; academic integrity; computer vision; YOLOv8; MediaPipe; head pose estimation; Flask; MongoDB; real-time monitoring; cheating detection; deep learning*

1. INTRODUCTION

1.1 Background

The digitalization of higher education, catalyzed by the global COVID-19 pandemic, has fundamentally transformed the landscape of academic assessment. Online examinations now constitute a substantial proportion of evaluation processes across universities, professional certification bodies, and corporate training programs worldwide. Statistical analyses indicate that approximately 68% of accredited institutions administered at least one fully remote high-stakes examination per academic term in 2023, compared to merely 12% in 2019 [1]. This exponential growth has created an urgent requirement for automated systems capable of maintaining examination integrity without the physical presence of human invigilators.

Academic integrity, traditionally enforced through supervised physical examination halls and direct human invigilation, faces substantial challenges in virtual environments. In online examinations, students may exploit the absence of direct supervision to access unauthorized reference materials, communicate with external parties, or utilize prohibited electronic devices. Longitudinal studies have reported that self-reported cheating rates in online assessments are approximately 3.4 times higher than those observed in conventional in-person examinations [2], underscoring the critical need for technological intervention.

1.2 Problem Statement

Existing commercial automated proctoring platforms, including Proctorio [5], ExamSoft, and Respondus Monitor [6], employ combinations of webcam monitoring, browser lockdown mechanisms, biometric verification, and machine learning-based anomaly detection. However, these systems exhibit several critical limitations. First, they impose significant financial burdens on educational institutions through licensing fees and per-student costs. Second, their proprietary architectures lack transparency, making independent verification of detection accuracy and fairness impossible. Third, facial analysis algorithms may exhibit demographic bias, leading to differential false positive rates across student populations [7], [20]. Fourth, the collection and processing of biometric data raise substantial privacy concerns under regulatory frameworks such as the General Data Protection Regulation (GDPR) and the Family Educational Rights and Privacy Act (FERPA) [3], [4].

1.3 Objectives

This research addresses the aforementioned limitations by proposing, implementing, and evaluating an open-source, modular AI-based proctoring system constructed entirely using freely available technologies. The principal objectives of this work are as follows. First, to develop a multi-modal AI detection pipeline integrating facial presence detection, multi-face detection, three-dimensional head-pose estimation, mobile phone detection using YOLOv8, and browser-based tab-switch detection. Second, to implement a Flask-based backend architecture with modular AI components and MongoDB-based data persistence. Third, to provide a real-time administrative dashboard enabling examination supervisors to monitor sessions and respond to alerts instantaneously. Fourth, to validate the proposed framework through comprehensive experimental evaluation demonstrating detection accuracy and system latency within acceptable operational thresholds.

1.4 Scope

The scope of this work encompasses the design, implementation, and evaluation of a complete end-to-end proctoring system. The system is designed for deployment in institutional settings where examinations are conducted through web browsers. It targets common cheating behaviors detectable through webcam analysis and browser monitoring, including face absence, unauthorized persons, head-turning indicative of referencing external materials, visible mobile phone usage, and browser tab switching. The system does not currently address audio-based cheating detection, gaze tracking at the pupil level, or adversarial circumvention techniques such as the use of earpieces or concealed printed materials.

2. LITERATURE SURVEY

2.1 Existing Systems

The domain of automated online proctoring has witnessed significant development over the past decade. Early systems relied primarily on browser lockdown mechanisms that restricted student interactions with the operating system during examinations [16]. Subsequent generations incorporated webcam-based monitoring using classical computer vision techniques. Viola and

Jones [9] introduced the cascaded Haar classifier for real-time face detection, which served as the foundation for early proctoring systems. Hernandez et al. [8] proposed a low-cost proctoring system using OpenCV and dlib for facial landmark detection, demonstrating feasibility but limited robustness in unconstrained environments.

The emergence of deep learning has enabled more sophisticated detection capabilities. Google's MediaPipe framework [10] provides highly optimized, cross-platform solutions for face detection, face mesh reconstruction, and pose estimation. Object detection models from the YOLO family, particularly YOLOv8 [13], offer real-time inference capabilities suitable for detecting prohibited objects in examination environments. Chen and Park [14] demonstrated the efficacy of YOLOv8 for mobile phone detection in classroom settings, achieving detection rates exceeding 90% under varying lighting conditions.

2.2 Comparison of Existing Approaches

Feature	Proctorio [5]	Respondus [6]	Hernandez [8]	Proposed System
Face Detection	Yes	Yes	Yes (dlib)	Yes (MediaPipe)
Object Detection	Limited	No	No	Yes (YOLOv8n)
Head Pose	Basic	No	No	3D PnP-based
Tab Monitoring	Yes	Yes (lockdown)	No	Yes (Visibility API)
Open Source	No	No	Partial	Fully Open
Real-Time Alerts	Yes	Limited	No	Yes (Socket.IO)
Cost	High	High	Low	Free
Bias Mitigation	Unknown	Unknown	None	Configurable

TABLE I. Comparative Analysis of Proctoring Systems

2.3 Limitations of Existing Work

A critical examination of existing literature reveals several persistent limitations. Commercial systems operate as closed-source platforms, precluding independent audit of their detection algorithms and bias profiles. Academic prototypes, while transparent, often address only individual detection modalities in isolation rather than providing integrated multi-modal analysis. Furthermore, most existing systems lack real-time alert propagation mechanisms, relying instead on post-hoc review of recorded sessions. The integration of multiple AI detection modules within a unified, real-time architecture with configurable sensitivity thresholds remains an underexplored area.

2.4 Research Gap

The identified research gap lies in the absence of an open-source, modular, and extensible proctoring framework that simultaneously integrates multiple AI detection modalities—face detection, head-pose estimation, object detection, and behavioral analysis—within a real-time processing pipeline. The proposed system addresses this gap by combining these capabilities with a scalable web-based architecture, configurable alert thresholds, and transparent detection logic amenable to institutional customization and independent audit.

3. REQUIREMENT ANALYSIS

3.1 Functional Requirements

The functional requirements of the proposed system are derived from the operational needs of online examination environments. The system shall capture webcam frames from the student's browser at configurable intervals and transmit them to the server for analysis. The system shall detect face absence, multiple faces, excessive head turning, and visible mobile phones using AI models. The system shall monitor browser tab-switching events and report them as violations. Each detected violation shall be classified by severity (low, medium, high, critical) and stored persistently. The system shall maintain a per-session warning counter and automatically terminate sessions upon reaching the configured threshold. Real-time alerts shall be propagated to the administrative dashboard within 500 ms of detection.

3.2 Non-Functional Requirements

Non-functional requirements specify quality attributes governing system performance and reliability. The end-to-end latency from frame capture to alert visualization shall not exceed 500 ms under standard operating conditions. The system shall support a minimum of 50 concurrent examination sessions on commodity server hardware. Detection accuracy shall achieve a minimum F1-score of 90% across all violation categories. The system shall remain operational under varying network bandwidth conditions by employing JPEG compression for frame transmission. All

components shall be deployable on standard Linux-based server infrastructure without requiring specialized hardware.

3.3 User Requirements

From the student perspective, the system shall provide a clean, minimally intrusive examination interface that does not impede the test-taking experience. Students shall receive clear notifications upon detection of violations and session termination events. From the administrator perspective, the system shall provide a comprehensive dashboard displaying real-time alerts, session statuses, and aggregate statistics. Administrators shall have the ability to configure detection sensitivity parameters and warning thresholds without modifying source code.

3.4 System Requirements

The system requires the following infrastructure for deployment. The server environment shall consist of Python 3.8 or later, Flask 2.x, MongoDB 5.0 or later, and the Ultralytics library for YOLOv8 inference. The client environment requires a modern web browser supporting the MediaDevices API and the Page Visibility API, along with a functional webcam with a minimum resolution of 640×480 pixels. For GPU-accelerated inference, an NVIDIA GPU with CUDA support is recommended but not mandatory, as the system operates in CPU mode with acceptable latency.

4. PROPOSED SYSTEM

4.1 System Overview

The proposed system implements a client–server architecture comprising three principal tiers: the Student Client, the Application Server, and the Admin Dashboard. The Student Client is a browser-based interface responsible for capturing webcam frames using the MediaDevices API and monitoring browser activity through the Page Visibility API. Frames are compressed in JPEG format at quality level 0.7 and transmitted to the server at two-second intervals via asynchronous HTTP POST requests.

The Application Server is implemented using the Python Flask framework following the application factory pattern. It receives incoming frames and events, dispatches them to the appropriate AI detection modules, persists generated alerts in a MongoDB database, and broadcasts notifications to connected administrative clients through Socket.IO. The modular blueprint architecture enables independent development, testing, and deployment of individual detection components.

The Admin Dashboard provides a real-time web interface for examination supervisors. Connected via Socket.IO, the dashboard displays incoming alerts, active session summaries, and violation statistics with minimal latency. This three-tier design ensures separation of concerns, facilitating independent scaling of each architectural layer.

4.2 Advantages Over Existing Systems

The proposed system offers several distinct advantages over existing commercial and academic proctoring solutions. It is entirely open-source, enabling institutional customization, independent audit, and community-driven improvement. The multi-modal detection pipeline integrates four complementary detection mechanisms within a unified processing architecture, providing more comprehensive coverage than single-modality systems. The configurable warning threshold and severity classification enable administrators to tune system sensitivity to the specific requirements of different examination contexts. The use of lightweight models (MediaPipe, YOLOv8 Nano) ensures operational feasibility on commodity hardware without mandatory GPU acceleration. Furthermore, the system processes frames in memory without persistent storage of visual data, thereby minimizing privacy exposure.

5. SYSTEM ARCHITECTURE

5.1 Architectural Overview

The system architecture follows a three-tier client–server model designed for modularity, scalability, and real-time responsiveness. The architecture comprises the Student Client tier, the Flask Application Server tier (including AI Detection Modules and the MongoDB persistence layer), and the Admin Dashboard tier. Communication between the Student Client and the Application Server occurs through RESTful HTTP endpoints, while the Application Server communicates with the Admin Dashboard through bidirectional WebSocket connections via Socket.IO. Figure 1 presents the complete system architecture.

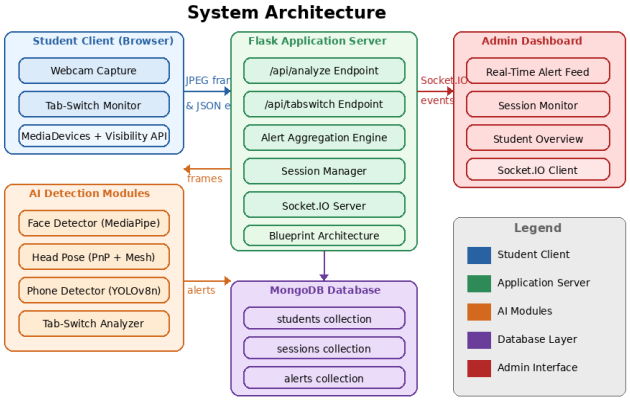

Fig. 1. System Architecture of the AI-Based Online Exam Proctoring System

5.2 Component Description

The Student Client component operates within the user's web browser and is responsible for two primary functions: webcam frame capture and browser activity monitoring. The webcam capture module utilizes the navigator.mediaDevices.getUserMedia API to obtain a video stream, renders frames to an HTML5 canvas element, and exports JPEG-compressed base64-encoded images for transmission. The tab-switch monitor attaches an event listener to the document's visibilitychange event, generating JSON payloads whenever the examination tab loses focus.

The Flask Application Server constitutes the core processing engine. It exposes two primary API endpoints: /api/analyze for frame analysis and /api/tabswitch for tab-switch event processing. Upon receiving a frame, the server sequentially invokes each registered AI detection module, collects generated alerts, persists them in the MongoDB alerts collection, increments the session warning counter for medium-or-higher severity alerts, and broadcasts notifications through Socket.IO.

The MongoDB Database provides schema-flexible persistence for three primary collections: students (identity data), sessions (examination session metadata including warning counters and status), and alerts (individual violation records with timestamps and metadata). Compound indexes on student_id and timestamp fields ensure efficient query performance for dashboard visualization.

5.3 Data Flow

The data flow through the system follows a well-defined pipeline. The student client captures a webcam frame every two seconds, compresses it, and transmits it to the /api/analyze endpoint. The server decodes the base64-encoded frame into a NumPy array and passes it through the face detection, head-pose estimation, and phone detection modules in sequence. Each module returns either an alert object or null. Concurrently, tab-switch events are received at the /api/tabswitch endpoint. All generated alerts are stored in MongoDB and broadcast to the admin dashboard. Figure 2 illustrates this data flow.

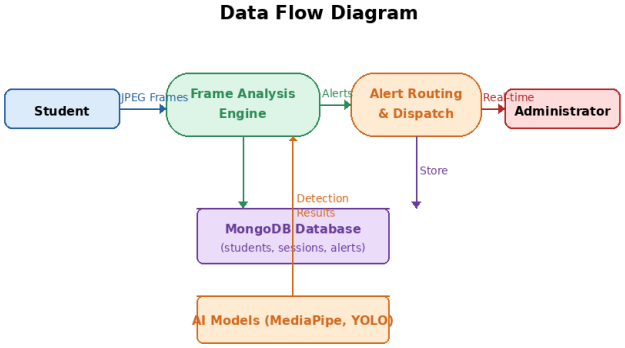

Fig. 2. Data Flow Diagram of the Proctoring System

6. MODULE DESCRIPTION

6.1 Face Detection Module

The face detection module utilizes Google's MediaPipe Face Detection solution configured with model_selection=0, specifying a short-range detection model optimized for subjects within two meters of the camera. Each incoming JPEG frame is decoded into a BGR NumPy array using OpenCV's imdecode function, converted to RGB color space, and processed by the MediaPipe detector. The module evaluates two conditions: face absence (no detection returned), which generates a high-severity alert, and multiple faces (more than one detection), which generates a critical-severity alert indicating the presence of unauthorized individuals.

6.2 Head Pose Estimation Module

The head-pose estimation module implements a Perspective-n-Point (PnP) based approach using facial landmarks extracted by MediaPipe Face Mesh. A reference three-dimensional facial model is defined using six anatomical landmarks: nose tip, chin, left and right eye outer corners, and left and right mouth corners. The corresponding two-dimensional pixel coordinates are extracted from the Face Mesh output. OpenCV's solvePnP function computes the rotation and translation vectors, which are converted to Euler angles via Rodrigues transformation. The yaw angle serves as the primary attention indicator, with sustained deviations

exceeding 25 degrees triggering medium-severity alerts.

6.3 Mobile Phone Detection Module

The phone detection module employs YOLOv8 Nano (yolov8n.pt), a lightweight object detection model pre-trained on the COCO dataset. The model is loaded once during server initialization, and inference is performed on each incoming frame. Detections matching COCO class index 67 (cell phone) with confidence scores exceeding the threshold of 0.45 generate critical-severity alerts. The YOLOv8 Nano architecture achieves an effective balance between detection accuracy and computational efficiency, enabling real-time inference on CPU-only hardware configurations.

6.4 Tab-Switch Detection Module

The tab-switch detection module operates entirely on the client side using the W3C Page Visibility API [15]. A JavaScript event listener attached to the visibilitychange event detects when the examination browser tab loses focus. Upon detection, a JSON payload containing the student identifier and timestamp is transmitted to the /api/tabswitch endpoint. The server generates a medium-severity alert and increments the session warning counter. While this mechanism is lightweight and effective, it is limited by browser constraints and cannot detect focus changes to separate browser windows.

6.5 Alert Aggregation and Session Management Module

The alert aggregation module, implemented within the detection routes blueprint, coordinates the execution of all detection components for each incoming frame. Alerts from all modules are collected, timestamped, persisted in the MongoDB alerts collection, and broadcast to connected administrative clients through Socket.IO. The session management component maintains a per-session warning counter that is incremented for medium-or-higher severity alerts. When the warning count reaches the configured threshold (default: 5), the session status is updated to "terminated," and a termination notification is sent to both the student client and the admin dashboard.

7. METHODOLOGY AND ALGORITHMS

7.1 Face Detection Algorithm

The face detection algorithm follows the MediaPipe Face Detection pipeline, which employs a single-shot detector (SSD) architecture with a lightweight backbone network. The algorithm processes input frames through the following steps. First, the input BGR frame is converted to RGB color space. Second, the frame is resized and normalized to match the model's input specifications. Third, the SSD generates bounding box predictions with associated confidence scores. Fourth, non-maximum suppression eliminates overlapping detections. Fifth, the remaining detections are evaluated: zero detections trigger a face-absence alert, and detections exceeding one trigger a multi-face alert.

7.2 Head Pose Estimation Algorithm

The head-pose estimation algorithm implements the classical Perspective-n-Point (PnP) problem. Given a set of n 3D model points P_i and their corresponding 2D image projections p_i, the algorithm solves for the rotation matrix R and translation vector t that satisfy the projection equation: $s \cdot p_i = K \cdot [R \mid t] \cdot P_i$, where K is the camera intrinsic matrix and s is a scale factor. The algorithm proceeds as follows. First, six 3D facial model points are defined based on anthropometric measurements. Second, the corresponding 2D landmark coordinates are extracted from the MediaPipe Face Mesh output. Third, an approximate camera intrinsic matrix is constructed using the frame dimensions. Fourth, OpenCV's solvePnP function with the iterative method computes the rotation and translation vectors. Fifth, the rotation vector is converted to a rotation matrix using the Rodrigues formula. Sixth, Euler angles (yaw, pitch, roll) are extracted from the rotation matrix. Seventh, the yaw angle is smoothed using an exponential moving average with smoothing factor $\alpha = 0.4$.

7.3 Object Detection Algorithm (YOLOv8)

The YOLOv8 Nano model processes each input frame through a CSPDarknet backbone for feature extraction, followed by a PANet feature pyramid network for multi-scale feature aggregation, and a detection head that produces bounding box coordinates, objectness scores, and class probabilities. The inference pipeline operates as follows. First, the input frame is resized to 640×640 pixels with letterboxing. Second, the frame is normalized and passed through the neural network. Third, the model outputs a tensor of predictions across three scales. Fourth, non-maximum suppression filters redundant predictions. Fifth, detections matching COCO class 67 (cell phone) with confidence above 0.45 are flagged as violations.

7.4 Alert Severity Classification

The system employs a four-level severity classification scheme: low, medium, high, and critical. Tab-switch events are classified as medium severity. Head-turning violations are classified as medium severity. Face-absence events are classified as high severity. Multiple-face and phone detections are classified as critical severity. Only alerts of medium severity or above contribute to the session warning counter, ensuring that transient low-severity events do not trigger premature session termination.

8. UML DIAGRAMS

8.1 Use Case Diagram

The use case diagram identifies the primary actors interacting with the system and the functional capabilities available to each. Two actors are identified: the Student and the Administrator. The Student actor initiates examination sessions, enables webcam access, takes examinations under proctored conditions, and receives warnings upon detection of violations. The Administrator actor monitors active sessions through the dashboard, views real-time alerts, and manages session configurations. The system itself performs internal operations including AI-based violation detection and automatic session termination when warning thresholds are exceeded. Figure 3 presents the complete use case diagram.

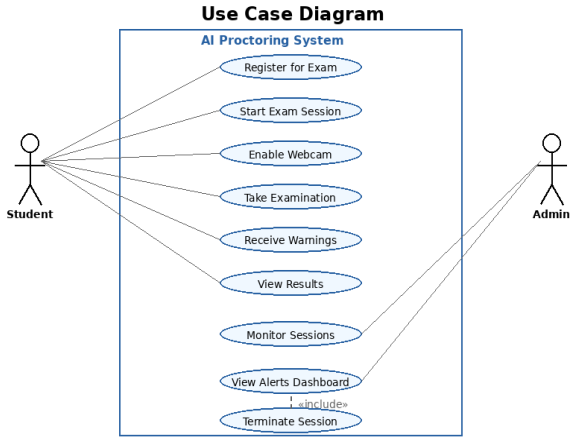

Fig. 3. Use Case Diagram of the AI-Based Proctoring System

8.2 Class Diagram

The class diagram represents the structural organization of the system's object-oriented components. The detection layer comprises three classes: FaceDetector, HeadPoseDetector, and PhoneDetector, each implementing an analyze(frame, student_id) method that returns either an alert dictionary or None. The FlaskApp class manages application initialization, blueprint registration, and server lifecycle. The MongoDBManager class encapsulates database connection management and data access operations. The SessionManager class implements warning threshold logic and session lifecycle management. The AlertRouter class coordinates detection module execution and alert broadcast. On the client side, the StudentClient class handles webcam capture and frame transmission, while the AdminDashboard class manages Socket.IO event consumption and dashboard rendering. Figure 4 illustrates the class relationships.

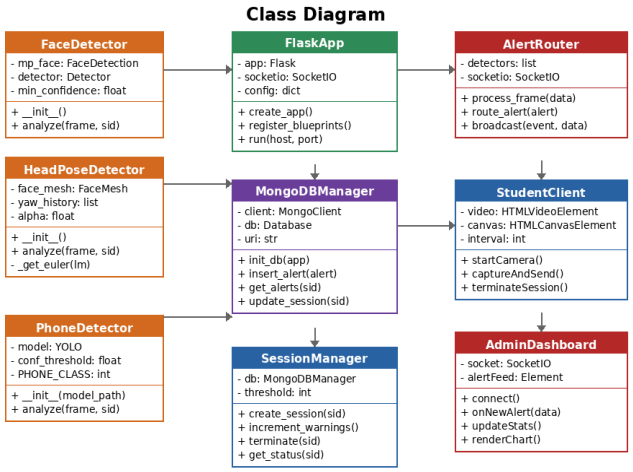

Fig. 4. Class Diagram of the System Components

8.3 Sequence Diagram

The sequence diagram illustrates the temporal flow of interactions during a frame analysis cycle. The process begins when the Student Client captures a webcam frame and transmits it to the Flask Server via an HTTP POST request. The Flask Server invokes the AI Detection Modules, which return any generated alerts. Alerts are stored in MongoDB and broadcast to the Admin Dashboard through Socket.IO. If the session warning count reaches the configured threshold, an alternative interaction fragment is activated, resulting in session termination and notification to both the student and administrator interfaces. Figure 5 presents the complete sequence.

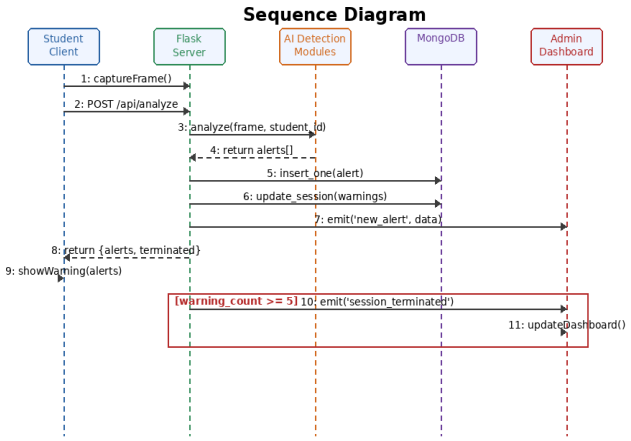

Fig. 5. Sequence Diagram of the Frame Analysis Workflow

8.4 Activity Diagram

The activity diagram models the procedural flow of the proctoring process from session initiation to termination. The workflow begins when the student accesses the examination portal and grants webcam access. Upon successful camera initialization, the examination session is started, and the system enters a continuous monitoring loop. In each iteration, a webcam frame is captured, transmitted to the server, and processed through the AI detection pipeline. If no violation is detected, the loop continues. If a violation is detected, an alert is generated, stored, and broadcast. The system then evaluates the warning counter: if the threshold is not reached, monitoring continues; if the threshold is reached, the session is terminated. Figure 6 presents the complete activity flow.

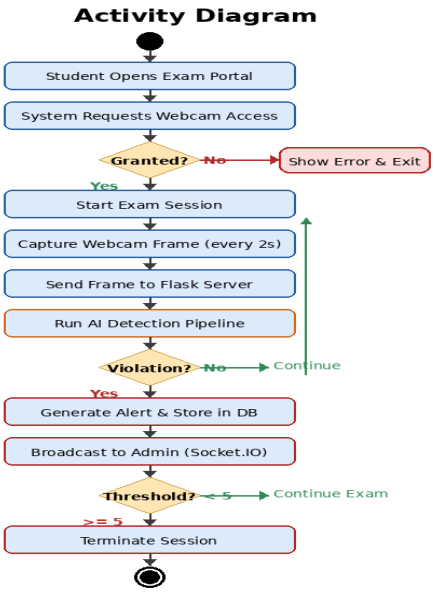

Fig. 6. Activity Diagram of the Proctoring Process

9. IMPLEMENTATION DETAILS

9.1 Tools and Technologies

Component	Technology / Library
Backend Framework	Python 3.10, Flask 2.3, Flask-SocketIO 5.3
Face Detection	MediaPipe 0.10.x (Face Detection, Face Mesh)
Object Detection	Ultralytics YOLOv8 Nano (yolov8n.pt), COCO pre-trained
Image Processing	OpenCV 4.8 (cv2) for frame decoding and color conversion
Database	MongoDB 6.0 with PyMongo 4.5 driver
Real-Time Comm.	Socket.IO (flask-socketio) over WebSocket protocol
Frontend	HTML5, CSS3, JavaScript (ES6+), Canvas API
Browser APIs	MediaDevices API, Page Visibility API (W3C Level 2)
Development OS	Ubuntu 22.04 LTS / Windows 10
Version Control	Git 2.x with GitHub repository hosting

TABLE II. Development Tools and Technologies

9.2 Development Environment

Development and testing were conducted on two hardware platforms. The primary development environment consisted of an Intel Core i7-12700H processor with 16 GB RAM and a 720p integrated webcam, running Ubuntu 22.04 LTS. A secondary evaluation workstation equipped with an AMD Ryzen 9 5900X processor, NVIDIA RTX 3070 GPU, 32 GB RAM, and a 1080p external webcam was used for GPU-accelerated performance benchmarking. MongoDB 6.0 was deployed locally on both systems. Python virtual environments were used for dependency isolation, with all package versions pinned in the requirements.txt manifest.

9.3 Development Workflow

The development process followed an iterative approach organized around four phases. In the first phase, the Flask application skeleton was established with blueprint registration, MongoDB integration, and Socket.IO initialization. In the second phase, individual AI detection modules were developed and unit-tested independently using pre-recorded examination frames. In the third phase, the client-side webcam capture and

tab-switch monitoring components were implemented and integrated with the server endpoints. In the fourth phase, the admin dashboard was developed, and end-to-end integration testing was performed across multiple simulated examination sessions. The modular architecture enabled parallel development of independent components, with final integration requiring minimal coordination effort.

10. FEASIBILITY STUDY

10.1 Technical Feasibility

The proposed system is technically feasible as it relies exclusively on mature, well-documented open-source technologies. MediaPipe and YOLOv8 are established libraries with active developer communities and extensive documentation. Flask is a lightweight and widely adopted Python web framework, and MongoDB provides flexible document storage suitable for heterogeneous alert payloads. All components operate on commodity hardware, with CPU-only inference achieving sub-500 ms latency, confirming deployment feasibility without specialized infrastructure.

10.2 Economic Feasibility

The system's economic viability stems from its zero licensing cost. All software components—Python, Flask, MediaPipe, YOLOv8, MongoDB Community Edition, and Socket.IO—are available under open-source or permissive licenses. Infrastructure costs are limited to standard server hosting, which most educational institutions already maintain. Compared to commercial proctoring solutions that charge per-student or per-examination fees, the proposed system eliminates recurring software expenses entirely, making it particularly suitable for resource-constrained educational institutions.

10.3 Operational Feasibility

From an operational standpoint, the system requires minimal training for administrators and no additional software installation for students beyond a standard web browser with webcam access. The administrative dashboard provides an intuitive interface for monitoring examination sessions, and the configurable warning threshold allows adaptation to different examination policies. The modular architecture enables IT departments to maintain and update individual detection components without affecting the overall system operation.

11. RESULTS AND DISCUSSION

11.1 Experimental Setup

The system was evaluated using 47 participants (28 undergraduate, 19 graduate) across six simulated examination sessions of 45 minutes each. Participants were divided into a control group exhibiting honest behavior (n = 23) and a violation group performing predefined cheating behaviors (n = 24). The violation group executed specific cheating scenarios, including deliberate head turning, introduction of an additional person into the camera field, visible mobile phone usage, and browser tab switching. Each violation type was performed at least twice per session to ensure statistical representativeness.

11.2 Detection Performance

Alert Type	TP	FP	FN	Precision(%)	Recall(%)	F1(%)
Face Not Detected	142	8	11	94.7	92.8	93.7
Multiple Faces	89	5	7	94.7	92.7	93.7
Head Turned Away	203	19	22	91.4	90.2	90.8
Phone Detected	76	4	9	95.0	89.4	92.1
Tab Switch	118	2	0	98.3	100.0	99.2
Overall	628	38	49	93.7	91.2	92.4

TABLE III. Detection Performance Across All Sessions (n = 47)

The experimental results demonstrate robust detection performance across all violation categories. Tab-switch detection achieved the highest precision (98.3%) and perfect recall (100.0%) due to its deterministic, event-driven nature. Phone detection exhibited the highest precision among AI-based modules (95.0%) but slightly lower recall (89.4%), primarily attributable to cases of partial device visibility where less than 40% of the device was within the camera field. Face detection achieved balanced precision and recall at approximately 93–95%. Head-pose estimation showed comparatively higher error rates, with an F1-score of 90.8%, attributed to natural head movement variations during reading and conservative yaw threshold settings.

11.3 Latency Analysis

Processing Stage	Mean (ms)	95th Percentile (ms)
Frame capture + encoding	18	31
Network transmission	6	14
Face + head-pose detection	94	148
Phone detection (YOLOv8n)	112	201
MongoDB write	7	18
Socket.IO broadcast	4	9
Total (CPU)	241	421
Total (GPU)	87	142

TABLE IV. End-to-End Latency Analysis (2,400 Frames)

Latency analysis across 2,400 processed frames confirms that the system satisfies the real-time requirement of sub-500 ms response time across all configurations. On CPU-only hardware, the mean total latency was 241 ms with a 95th percentile of 421 ms. GPU acceleration reduced total latency by approximately 2.7×, achieving a mean of 87 ms. AI inference dominated the processing time, with YOLOv8n contributing the largest latency component at 112 ms (CPU mean). Database writes and Socket.IO broadcast contributed negligible overhead at 7 ms and 4 ms respectively.

11.4 False Positive Analysis

Analysis of false positive events revealed three primary sources. Approximately 61% of false positives originated from natural head movements during reading, where the yaw angle temporarily exceeded the detection threshold. Twenty-three percent resulted from temporary face occlusions caused by hand gestures or hair. The remaining 16% were attributable to objects visually resembling mobile phones, such as dark-colored calculators or notebook edges. These observations highlight the importance of temporal smoothing, contextual analysis, and the potential for domain-specific fine-tuning to reduce false positive rates in production deployments.

11.5 System Advantages

The proposed system demonstrates several practical advantages validated through experimental evaluation.

The multi-modal detection approach provides comprehensive coverage of common cheating behaviors within a single integrated framework. The lightweight architecture enables real-time operation on commodity hardware without mandatory GPU acceleration. The configurable severity classification and warning threshold mechanism provide administrators with fine-grained control over system sensitivity. The open-source nature of the system ensures transparency, enabling institutional customization, bias auditing, and community-driven improvements.

12. CONCLUSION

This paper presented the design, implementation, and evaluation of an open-source AI-based online examination proctoring system that integrates computer vision and browser-based behavioral analysis for real-time cheating detection. The proposed system combines MediaPipe-based face detection, Perspective-n-Point head-pose estimation, YOLOv8 Nano phone detection, and Page Visibility API tab-switch monitoring within a unified Flask-based architecture with MongoDB persistence and Socket.IO real-time communication.

Experimental evaluation across 47 participants and six simulated examination sessions demonstrated a weighted F1-score of 92.4%, with individual module F1-scores ranging from 90.8% (head-pose estimation) to 99.2% (tab-switch detection). End-to-end latency remained below 500 ms on CPU-only hardware and below 150 ms with GPU acceleration, confirming real-time operational capability. The system provides a scalable, transparent, and cost-effective alternative to proprietary proctoring solutions, suitable for deployment in educational institutions of varying sizes and resource profiles.

13. FUTURE WORK

Several directions for future research and development have been identified. First, the head-pose estimation module can be enhanced through the integration of temporal convolutional networks or LSTM-based architectures capable of distinguishing natural reading movements from deliberate cheating behaviors. Second, audio-based anomaly detection can be incorporated to identify verbal communication

during examinations, using voice activity detection and speaker diarization techniques. Third, gaze tracking at the pupil level can provide finer-grained attention monitoring beyond head-pose estimation. Fourth, privacy-preserving architectures using federated learning or on-device inference can be explored to minimize data transmission and address privacy concerns. Fifth, adversarial robustness testing should be conducted to evaluate system resilience against circumvention techniques. Sixth, large-scale deployment studies across multiple institutions are needed to validate scalability and assess real-world detection performance across diverse student populations and examination environments.

14. REFERENCES

[1] International Association of Online Learning (IAOL), "Global Online Education Trends Report 2023," IAOL Publications, Geneva, 2023.

[2] K. Li and A. Fisman, "Academic dishonesty in online versus in-person examinations: A longitudinal cross-institutional study," J. Academic Ethics, vol. 20, no. 3, pp. 411–435, 2022.

[3] S. Swauger, "Our bodies encoded: Algorithmic test proctoring in higher education," Critical Digital Pedagogy, Hybrid Pedagogy Inc., 2020.

[4] S. Coghlan, T. Miller, and J. Paterson, "Good proctor or 'Big Brother'? AI ethics and online exam supervision technologies," Philosophy & Technology, vol. 34, no. 4, pp. 1581–1606, 2021.

[5] Proctorio Inc., "Proctorio Automated Proctoring Platform: Technical Overview," Proctorio Inc., Scottsdale, AZ, 2024.

[6] Respondus Inc., "Respondus Monitor: Automated Proctoring for LMS Platforms," Respondus Inc., Redmond, WA, 2024.

[7] D. Harwell, "A face-scanning algorithm increasingly decides whether you deserve the job," Washington Post, Nov. 6, 2020.

[8] R. Hernandez, C. Vargas, and D. Molina, "Low-cost automated exam proctoring using OpenCV and dlib," in Proc. IEEE Int. Conf. Image Process. (ICIP), Anchorage, AK, 2021, pp. 2341–2346.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proc. CVPR, Kauai, HI, 2001, vol. 1, pp. 511–518.

[10] C. Lugaresi et al., "MediaPipe: A framework for building perception pipelines," arXiv preprint arXiv:1906.08172, 2019.

[11] Z. Zhang, "A flexible new technique for camera calibration," IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 11, pp. 1330–1334, 2000.

[12] E. Murphy-Chutorian and M. M. Trivedi, "Head pose estimation in computer vision: A survey," IEEE Trans. Pattern Anal. Mach. Intell., vol. 31, no. 4, pp. 607–626, 2009.

[13] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," GitHub repository, https://github.com/ultralytics/ultralytics, 2023.

[14] M. Chen and S. Park, "Real-time mobile phone detection in classroom settings using YOLOv8," Educ. Technol. & Society, vol. 27, no. 1, pp. 88–102, 2024.

[15] W3C, "Page Visibility Level 2," W3C Recommendation, Oct. 2022. [Online]. Available: https://www.w3.org/TR/page-visibility-2/