ONLINE EXAM PROCTORING SYSTEM USING ARTIFICIAL INTELLIGENCE

A Mini-Project Dissertation submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

in

SOFTWARE ENGINEERING

Submitted by

ERRI SUVARNA

(24011D2527)

Under the Esteemed Guidance of

Dr. K. SANTHI SREE

Prof, Dept of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

UNIVERSITY COLLEGE OF ENGINEERING, SCIENCE & TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

KUKATPALLY, HYDERABAD, TELANGANA, INDIA-500085

2024-2026

DEPARTMENT OF INFORMATION TECHNOLOGY

UNIVERSITY COLLEGE OF ENGINEERING, SCIENCE & TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

KUKATPALLY, HYDERABAD, TELANGANA, INDIA-500085

2024-2026

CERTIFICATE

This is to certify that the dissertation entitled **"ONLINE EXAM PROCTORING SYSTEM USING ARTIFICIAL INTELLIGENCE"** is being submitted by **ERRI SUVARNA** bearing Roll No: **24011D2527**in partial fulfilment of the degree of **Master of Technology** in **"Software Engineering",** to the **Department of Information Technology**, **JNTUH UCESTH** is a record of Bonafide work carried out by her under our guidance and supervision.

The results embodied in this Mini project have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide

Dr. K. SANTHI SREE

Prof, Dept of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

UNIVERSITY COLLEGE OF ENGINEERING, SCIENCE & TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

KUKATPALLY, HYDERABAD, TELANGANA, INDIA-500085

2024-2026

CERTIFICATE

This is to certify that the dissertation entitled **"ONLINE EXAM PROCTORING SYSTEM USING ARTIFICIAL INTELLIGENCE"** is being submitted by **ERRI SUVARNA** bearing Roll No: **24011D2527**in partial fulfilment of the degree of **Master of Technology** in **"Software Engineering",** to the **Department of Information Technology**, **JNTUH UCESTH** is a record of Bonafide work carried out by her under our guidance and supervision.

The results embodied in this Mini project have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department

Dr. V. UMA RANI

Prof & Head, Dept of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

UNIVERSITY COLLEGE OF ENGINEERING, SCIENCE & TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

KUKATPALLY, HYDERABAD, TELANGANA, INDIA-500085

2024-2026

DECLARATION

I, **ERRI SUVARNA,** bearing Roll No: **24011D2527,** here by declare that report of Post Graduate Mini-Project Work entitled **"ONLINE EXAM PROCTORING SYSTEM USING ARTIFICIAL INTELLIGENCE"** which is being submitted to the **Department of Information Technology, JNTUH UCESTH** in partial fulfilment of the degree **Master of Technology** in **"Software Engineering", Department of Information Technology,** is a Bonafide report of work carried out by me.

The material contained in this report has not been submitted to any other University or Institute for the award of any degree or diploma.

ERRI SUVARNA

(24011D2527)

ACKNOWLEDGEMENT

I extend my sense of gratitude to my guide **Dr. K. SANTHI SREE**, **Professor, Dept of IT and Mrs. RENUKA DEVI** (Guest Faculty), JNTUH UCESTH, for all the timely support, valuable suggestions, constant guidance, encouragement and moral support during the period of Mini-Project and permitting me to undertake this project.

I extend my sense of gratitude to my guide **Mrs. RENUKA DEVI** (Guest Faculty), JNTUH UCESTH, for all the timely support, valuable suggestions, constant guidance, encouragement and moral support during the period of Mini-Project and permitting me to undertake this project.

I extend my deep sense of gratitude to the **Head of the Department, Dr. V. UMA RANI, Professor, Dept of IT,** JNTUHUCESTH, for all the timely support, valuable suggestions, constant guidance, encouragement and moral support during the period of Mini-Project and permitting me to undertake this project.

I would like to express my sincere thanks to **DR. G. VENKATA NARISMHA REDDY, PRINCIPAL**, JNTUH UCESTH, for providing the working facilities in college.

Finally, I would also like to thank all the faculty and staff of Department of Information Technology, who helped me for completing this Mini-Project.

With Sincere Regards,

ERRI SUVARNA

24011D2527

ABSTRACT

In recent years, online education and assessments have become increasingly common, especially during and after the COVID-19 pandemic. However, ensuring fairness and preventing cheating during online exams remains a major challenge. This project presents an intelligent Online Exam Proctoring System that uses artificial intelligence to monitor students in real-time through their webcam.

The system is developed using **Python Flask** for the backend and **SQLite** for data storage, while detection is handled using **MediaPipe**, **OpenCV**, and **YOLOv5**. It can detect behaviours such as frequent eye blinking, mouth movements (indicating talking), unusual head movements, and the presence of restricted objects like mobile phones or additional people. When suspicious activities are detected multiple times, the system automatically ends the exam and logs the behaviour.

This approach reduces the need for human invigilators and increases the integrity of remote assessments. The system is lightweight, user-friendly, and provides a secure platform for conducting online exams.

Keywords: Online Proctoring, AI-based Monitoring, Remote Exam, YOLOv5, MediaPipe, Face Detection, Flask Web App.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 OVERVIEW

This paper presents an attempt to solve the current issue of virtually invigilating students during their online examinations. The system is designed to be convenient, affordable, and easy to use from both the examinees and examiner's perspectives since it only requires having one inexpensive camera and a microphone. With the captured videos and audio, we extract features from the following basic components: phone detection, speech detection, text detection, gaze estimation, active window detection, and user verification. These features are then processed to acquire high-level features and then are used for detecting whether the examinee is cheating or not. This proctoring solution was built which not only had audio and video functionalities but also had a portal relating to both the examinee and examiner part. It could also let the examiner proctor, multiple examinees, at a time.

1.2 MOTIVATION FOR THE PROJECT

Since the COVID-19 pandemic, every sector has been converted to online including the education industry. This resulted in the blossom of remote learning. Many online education institutions were opened. While many educational institutions have transformed from classroom teaching to using applications like Google Classroom or Microsoft Teams for online teaching and interaction purposes, but there has been no effective and actual solution to examinations. Many institutions have opted for simple take-home-assignments where cheating is almost a norm, while some just cancelled them completely. If the current circumstances are to be the future's norm there's an immense need for an effective solution.

The traditional approach to conducting exams involved students being physically present in a classroom or exam centre, with invigilators/supervisors monitoring their activities to prevent cheating. However, with the advent of technology and the rise of online education, conducting exams remotely has become increasingly popular. Due to this, creative solutions that can guarantee the validity of the exam procedure and discourage cheating are now required. The Online Test Proctoring System, which uses AI, offers a creative approach to these problems.

1.3 PROBLEM DEFINITION AND SCENARIOS

Many institutions are allowing students to give exams from home while being monitored by a camera remotely. This solution however is not at all scalable and infeasible at large scale due to the workforce required, even if it is implemented there's an extremely high possibility for the students to cheat. To ensure that students don't cheat and make their learning and testing experience valid, fair, and valuable, free and good proctoring software is needed. So, the right solution might be a proctoring system which can monitor the students using the laptop in-built webcam itself to achieve the target of software and hardware requirements being affordable and easily available. The software built to be scalable, and the examiners can monitor multiple students at once.

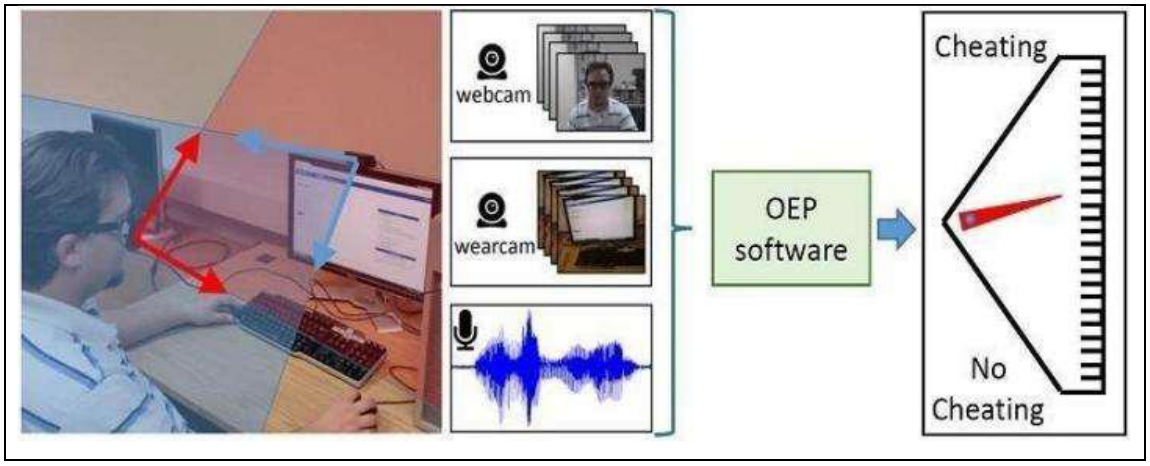


Fig: 1.1 Online Proctoring System

2. EXISTING SYSTEM

2.1 OBJECTIVE OF THE PROJECT WORK

The main objective of this project is to attempt to solve the problem of virtual proctoring of students during online examinations and to develop a standalone AI-powered system that ensures efficient and reliable monitoring. This work aims to provide a secure examination environment through real-time detection of suspicious behaviour such as unauthorized object presence, abnormal eye or head movements, and potential speaking activity using computer vision and deep learning techniques. The goal is to create a system that minimizes human intervention while maintaining academic integrity, offering a scalable and semi-automated proctoring solution suitable for remote learning scenarios.

2.2 EXISTING SYSTEM

In the current examination environment, traditional systems dominate, relying heavily on manual intervention for scheduling, supervision, and evaluation. These systems exhibit multiple limitations that hinder scalability, security, and accuracy, making them inefficient and unsuitable for modern academic and professional assessment needs. Manual supervision plays a dominant role in existing systems, where invigilators either physically monitor students or supervise through webcams. This approach is time-consuming, incurs high operational costs, and is limited by human capacity. The dependency on human involvement increases the likelihood of oversight and human error, thereby raising the risk of undetected malpractice during exams. Some institutions have adopted basic online platforms to conduct examinations; however, these lack sophisticated proctoring mechanisms. These platforms usually rely on simple screen recording and offer minimal or no AI-based

behaviour tracking or secure identity verification. Consequently, students can exploit such systems by switching tabs or using unauthorized resources, compromising the credibility and fairness of the exam process.

From a security and compliance perspective, the existing systems fall short due to the absence of integrated AI-based fraud detection and behavioural analysis tools. There are no real-time alerts or automated responses to suspicious activities, and the lack of robust security protocols further weakens exam integrity. These systems are unable to scale effectively for large-scale exams and offer no proactive mechanisms for identifying and mitigating cheating attempts in real time.

Thus, the current examination infrastructure is burdened with several critical limitations: it is inefficient due to manual verification and monitoring, expensive to operate at scale, vulnerable to cheating and fraud because of weak security measures, and lacks the real-time monitoring and automated alerting capabilities needed to ensure secure and fair examinations.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

In ai based proctoring system there are still several open problems and challenges. This includes:

1. Privacy Concerns: Striking a balance between effective proctoring and student privacy is paramount. While AI offers immense potential, minimizing intrusive data collection and safeguarding student information are crucial. This includes fostering transparency about data use and providing students with control over their information. The system should collect and store minimal data, anonymize it whenever possible, and encrypt it for security.

2. Algorithm Bias: Artificial intelligence algorithms are used to detect cheating may be biassed, resulting in unfair or inaccurate results, particularly for students

from diverse backgrounds. Mitigating algorithmic bias and ensuring equitable proctoring systems for all students are critical issues.

3. False Positives and Negatives: AI proctoring systems can sometimes produce false positives (incorrectly identifying cheating) or false negatives (failing to detect cheating). Reducing the occurrence of these errors and fine-tuning the algorithms to enhance accuracy is an ongoing challenge.

4. Complex Cheating Techniques: As students become more creative in their cheating methods, AI proctoring systems must continually adapt to detect new and evolving forms of dishonest behaviour.

5. User Experience: Ensuring a smooth and non-disruptive examination experience for students while using AI proctoring is an open issue. Striking a balance between effective monitoring and a user-friendly interface is essential to reduce student anxiety and maintain the integrity of the assessment process.

6. Student Acceptance: Convincing students of the necessity and fairness of AI proctoring can be difficult. Gaining student acceptance and addressing concerns about surveillance and fairness remains an ongoing challenge.

7. Cost-Effectiveness: Achieving a balance between the cost effectiveness of AI proctoring systems and the quality of proctoring services is an open problem. Reducing costs while maintaining or improving the accuracy and reliability of these systems is an ongoing challenge for educational institutions.

8. Feedback Mechanisms: Developing effective feedback mechanisms for students and instructors regarding proctoring outcomes is essential. Providing clear and constructive feedback on flagged behaviors can help students understand and address any issues. Addressing these open problems in AI-based proctoring systems will require collaborative efforts from researchers, educators, policymakers, and technology developers.

3. PROPOSED SYSTEM

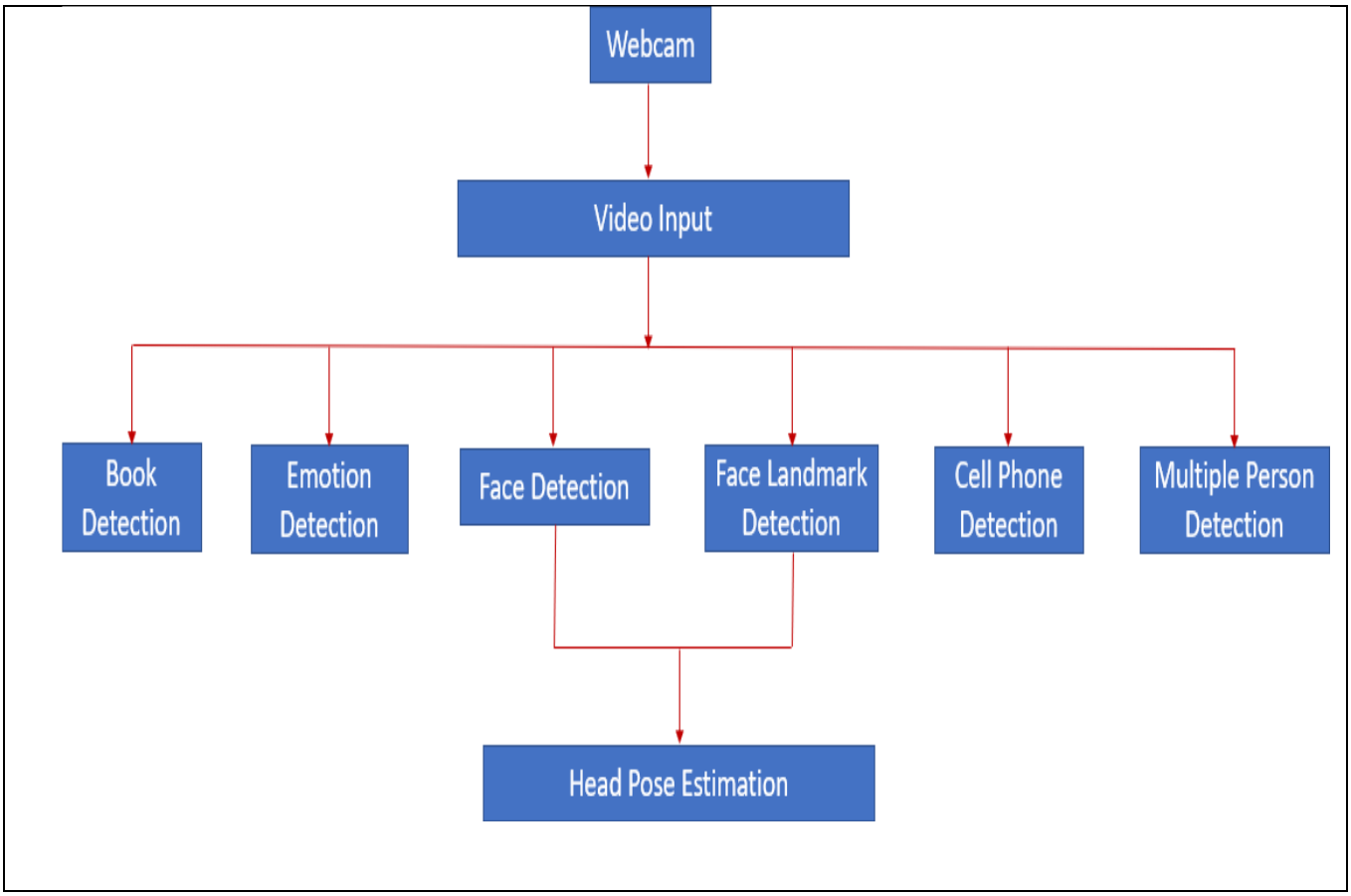


Fig: 3.1 Block Diagram of the Proposed System

In this project, we propose an AI-Based Online Exam Proctoring System designed implemented using Flask (Python backend), HTML/CSS/JavaScript (frontend), and real-time webcam-based monitoring with YOLOv5, Media Pipe, and OpenCV. The goal is to create a secure, automated online examination environment that continuously monitors student activity and flags suspicious behaviour without the need for human invigilators.

The system begins with student login and then activates the live webcam for the duration of the exam. Multiple AI modules run in parallel to detect various cheating behaviours and report them live on the screen. If the system identifies three or more violations, the exam is automatically terminated, and

the student is redirected to the result page. All detected events are also saved in a timestamped report file.

3.1. PROPOSED MODELS:

1. Blink Detection:

The system uses **MediaPipe Face Mesh** to detect the eye landmarks and compute the Eye Aspect Ratio (EAR). If the student's eyes remain closed beyond a specific threshold or if blinking is unusually frequent, it is flagged as a suspicious activity. This helps in identifying signs of drowsiness, distraction, or intentional attempts to avoid screen visibility.

2.Mouth Tracking:

By analyzing the position of lip landmarks using **MediaPipe**, the system determines whether the student's mouth is open or closed. An open mouth may suggest that the student is speaking, possibly receiving help or reading answers aloud, which violates examination integrity.

3.Head Pose Estimation:

The head pose is calculated by mapping facial landmarks into a 3D model and using vector geometry to estimate the direction of gaze. If the student consistently looks **away from the screen** (left, right, up, or down), it is considered a possible indication of cheating, such as viewing notes or devices placed outside the camera's view.

4. Object Detection (YOLOv5):

Identifies unauthorized items such as mobile phones, books, or extra persons in the frame using the YOLOv5 model integrated via PyTorch.

5.**Cheating Detection and Termination:**

The system maintains a **violation counter** for each student session. When any of the AI modules (blink, mouth, head pose, object) flags suspicious activity, the counter is incremented.

- If the counter reaches **three (3)**, the system automatically **terminates the exam session**, logs the behaviour, and redirects the student to a result or warning page.
- This ensures that repeated violations do not go unchecked and provides automated invigilation.

6. Live Detection Display:

A real-time webcam feed is displayed in the student's browser using HTML5 video and canvas elements. The detection results from all modules (blink, mouth, head pose, cheating status) are **overlaid on top of the webcam feed** as labels.

Example:

Blink: Yes | Mouth: Open | Head Pose: Right | Cheating: No

This enhances transparency and helps the student be aware of what the system is monitoring.

7. Detection Report Logging:

Every session generates a **timestamped log file** under the reports/ directory with the format:

reports/user_<id>_report.txt

These logs contain entries like:

[10:45:12] Blink: No, Mouth: Closed, Head Pose: Down, Cheating: No

[10:46:27] Blink: Yes, Mouth: Open, Head Pose: Left, Cheating: Yes

These files are used by admins for post-exam review and are an essential part of the audit trail.

8. SQLite Database:

The backend is powered by **SQLite**, a lightweight relational database system. It is used to:

- Store and manage student/admin login credentials.
- Maintain a question bank uploaded by the admin.

- Record final exam scores and timestamps.
- Ensure fast access and persistent storage without needing an external DBMS.

3.2. ADVANTAGES OF THE PROPOSED WORK

1. **Real-time Monitoring with AI**

   The system provides continuous real-time detection of cheating behavior using computer vision models like **MediaPipe** and **YOLOv5**, reducing the reliance on manual invigilation.
2. **Automated Exam Termination**

   Automatically ends the exam after a set number of violations (default is 3), improving security and integrity without human intervention.
3. **Lightweight Architecture**

   Uses **Flask** (a micro web framework) and **SQLite**, making it efficient and easily deployable even on basic systems or local servers.
4. **No Image or Video Storage**

   Focuses on privacy by avoiding storage of any webcam recordings or images. Only textual reports are saved, ensuring ethical compliance.
5. **Custom Report Generation**

   Every student gets a session-specific log (e.g., user_1_report.txt) that contains timestamped proctoring observations, helpful for post-exam analysis.
6. **User-Friendly Interface**

   Clean and intuitive frontend using HTML, CSS, and JS enables smooth navigation for both students and administrators.

4. SYSTEM ARCHITECTURE

The architecture of the **Online Exam Proctoring System using Artificial Intelligence** is designed to ensure secure, real-time monitoring of candidates during exams using advanced computer vision techniques. This system combines front-end interfaces, AI-based detection modules, and a backend database to provide a seamless and reliable examination platform. The architecture is composed of the following core components:

4.1. SYSTEM ARCHITECTURE DIAGRAM

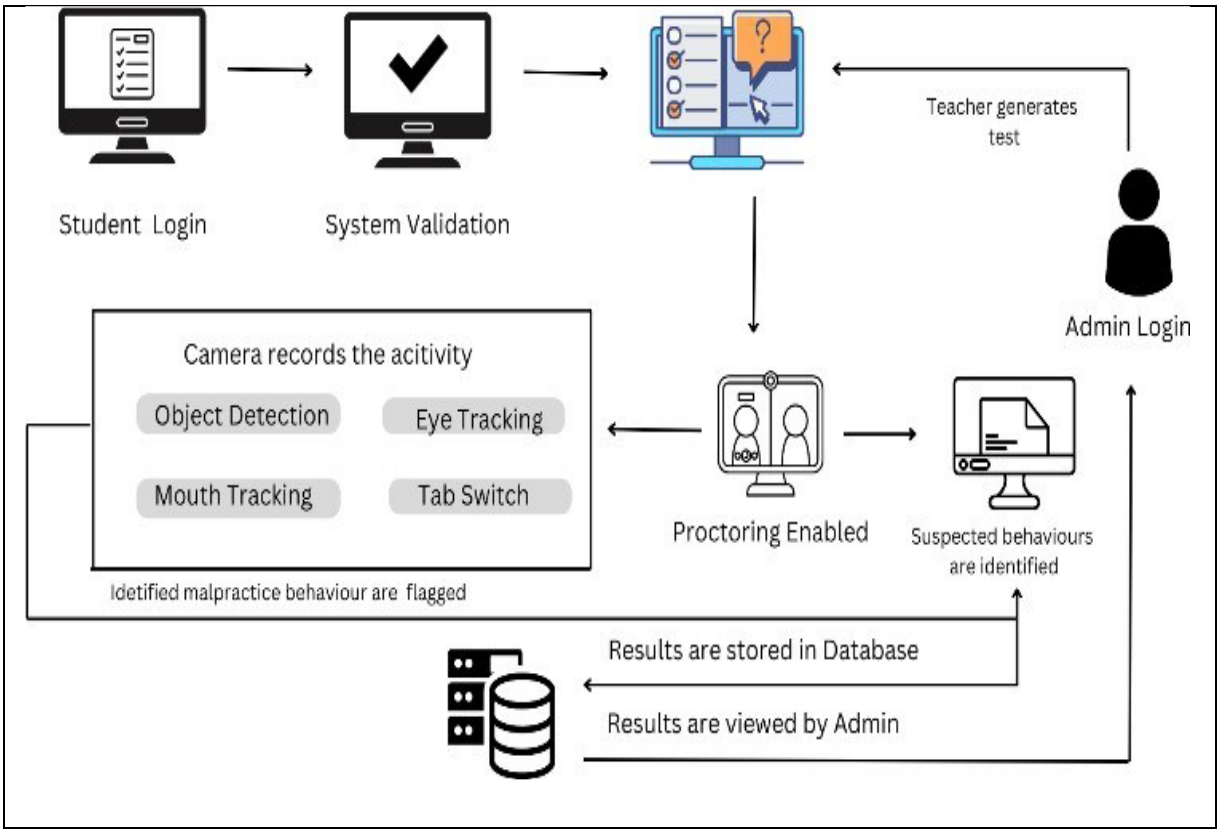


Fig 4.1: system architecture for ai based proctoring system

4.2. OVERVIEW OF THE ARCHITECTURE

The system consists of the following core components, categorized by frontend (client interface), backend (server logic), and AI processing units:

1.Client Interface (Frontend Technologies):

The **client interface** is uses HTML5, CSS3, and JavaScript to give students a smooth experience. It shows the live webcam feed, exam questions, and real-time detection labels like “Blink,” “Mouth,” “Head Pose,” and “Cheating” using a canvas overlay.

2.Backend (Flask Server and APIs):

The **backend** is developed using Python and the Flask web framework. It handles login, exam sessions, and result processing. It stores data in SQLite and generates proctoring reports. If a student breaks rules three times, the system ends the exam automatically.

3.Proctoring Engine (AI Detection Modules):

The AI engine uses MediaPipe, YOLOv5, and OpenCV for live monitoring. It detects blinking, mouth movement, head position, and suspicious activity through the webcam. Results like “Blink: Yes” or “Cheating: No” are sent for review.

4.3 FLOW OF OPERATION

1.Student Login: The system begins with the student logging into the platform through the web interface using their credentials. This step ensures that only registered students can access the examination environment.

2.System Validation: After login, the system performs validation checks to verify the student’s identity. This includes checking login credentials and optionally verifying the webcam and system capabilities (e.g., camera access, browser compatibility).

3.Test Generation by Admin: The administrator (teacher) logs in through a separate admin panel to upload, manage, and assign question papers to specific students. Admins can add MCQs and monitor all exam records. This module enables centralized control over exam content.

4.Exam Interface Activation: Once the exam is started, the student's interface displays the test along with a live webcam feed. The system then enables the proctoring module in the background, which monitors the student in real-time through various AI-based detections.

4.4 AI-BASED PROCTORING MODULE (CORE OF THE SYSTEM)

- **Object Detection:** Uses YOLOv5 to identify unauthorized objects like mobile phones, earphones, or multiple persons in the frame. If any restricted item is detected, a cheating flag is raised.
- **Eye Tracking / Blink Detection:** Implemented using MediaPipe Face Mesh, this module tracks the eye aspect ratio to detect abnormal blinking or eye closures, which may indicate distractions or cheating attempts.
- **Mouth Tracking:** Monitors if the mouth is frequently open, suggesting the possibility of speaking or receiving verbal help from others.
- **Head Pose Estimation:** Estimates the direction in which the student is looking (up, down, left, right). Repeated head movements away from the screen are considered suspicious.

4.5 CHEATING DETECTION AND FLAGGING

The **cheating detection module** is a core part of the AI-based proctoring system. It continuously analyzes the student's webcam feed using various detection algorithms such as **object detection (YOLOv5)**, **blink detection**, **mouth tracking**, and **head pose estimation**. If any suspicious activity is detected—such as the presence of a mobile phone, frequent blinking, mouth movements indicating speech, or the student looking away from the screen—it is marked as a **violation**. For every flagged detection, a **violation counter** is incremented.

The system is configured with a predefined threshold (typically 3 violations), beyond which the student is considered to have committed cheating. Once the violation counter reaches this limit, the exam is **automatically terminated by the backend**, and the student is **redirected to the result or warning page**. Throughout the exam, these detections are also **visually displayed on the student's interface**, showing labels like "Blink: Yes," "Head Pose: Right," or "Cheating: No" in real-time for transparency and awareness.

4.6 REPORT GENERATION AND RESULT HANDLING

In addition to live flagging, the system maintains a **time stamped log** of all detection events throughout the student's exam session. These logs are stored in a dedicated **text file** located in the reports/ directory, with filenames in the format user_<id>_report.txt. Each entry records the precise time and the results of each detection module—such as whether the student blinked, had their mouth open, moved their head, or was flagged for cheating.

For example:

[10:45:33] Blink: Yes, Mouth: Closed, Head Pose: Down, Cheating: No

[10:46:18] Blink: No, Mouth: Open, Head Pose: Left, Cheating: Yes

4.7 DATABASE LAYER

The backend of the system uses a **lightweight SQLite database** to manage all essential data. This includes:

- **User Data**: Contains login credentials, user roles (admin or student), and registration details.
- **Questions**: Stores the exam questions uploaded by the admin, typically in the form of multiple-choice questions (MCQs).
- **Results**: Logs exam scores, timestamps of submission, and the overall performance metrics of each student.

This database interacts with the **Flask backend APIs**, which perform operations such as login validation, question retrieval during the exam, and saving exam results.

4.8 USE CASE DIAGRAM

To visually represent the structure and workflows of the system, several UML diagrams are used. The Use Case Diagram outlines the major functionalities from the user's perspective. It shows how students interact with the system to sign up, log in, start exams, and submit them, while examiners create exams, assign access, and monitor results and violations.

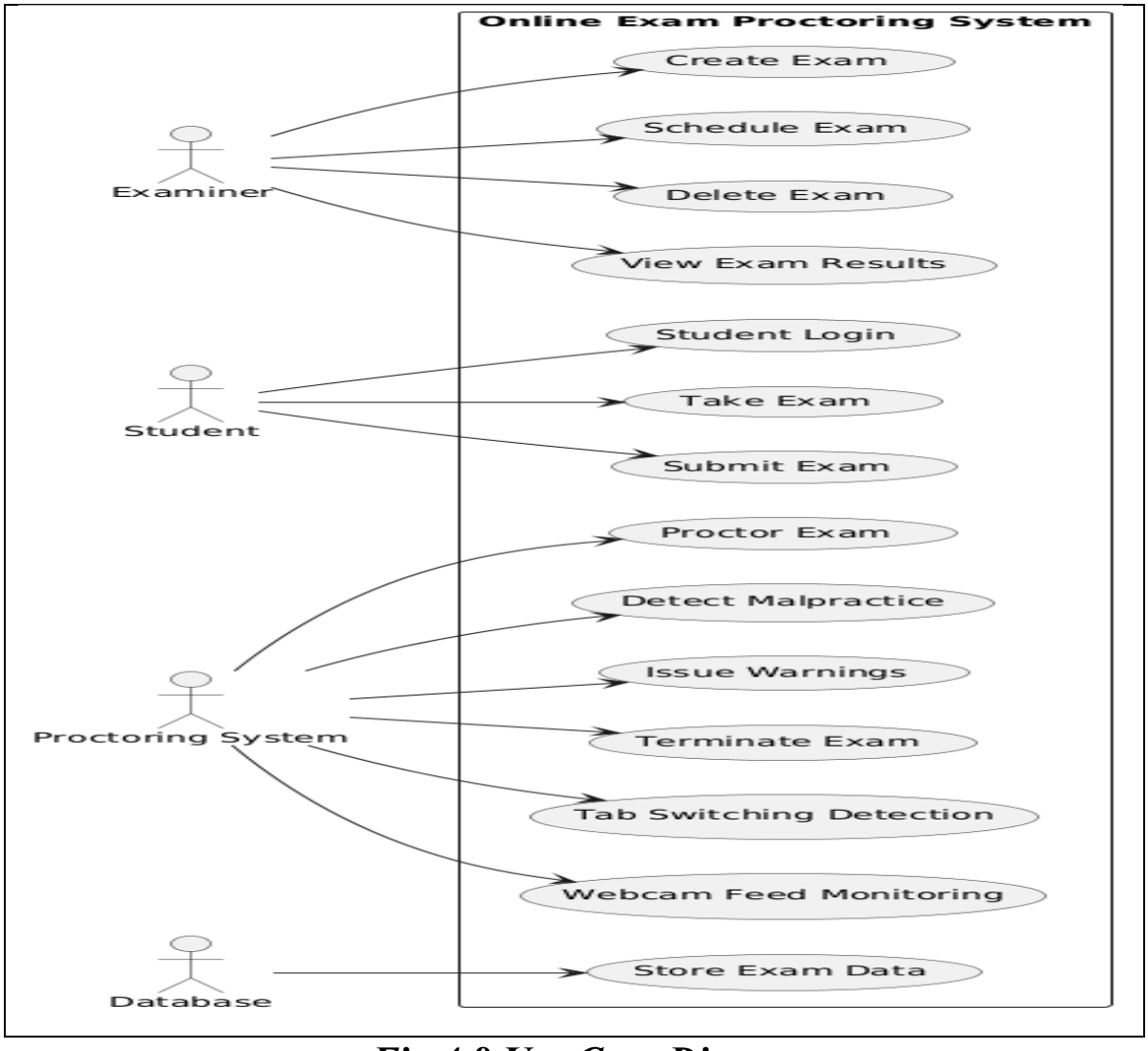


Fig 4.8 Use Case Diagram

4.9 SEQUENCE DIAGRAM

The Sequence Diagram captures the flow of events during a student's interaction with the system—from login and exam loading to real-time violation tracking and submission. It emphasizes the interactions between the client, server, and database in a time-ordered manner.

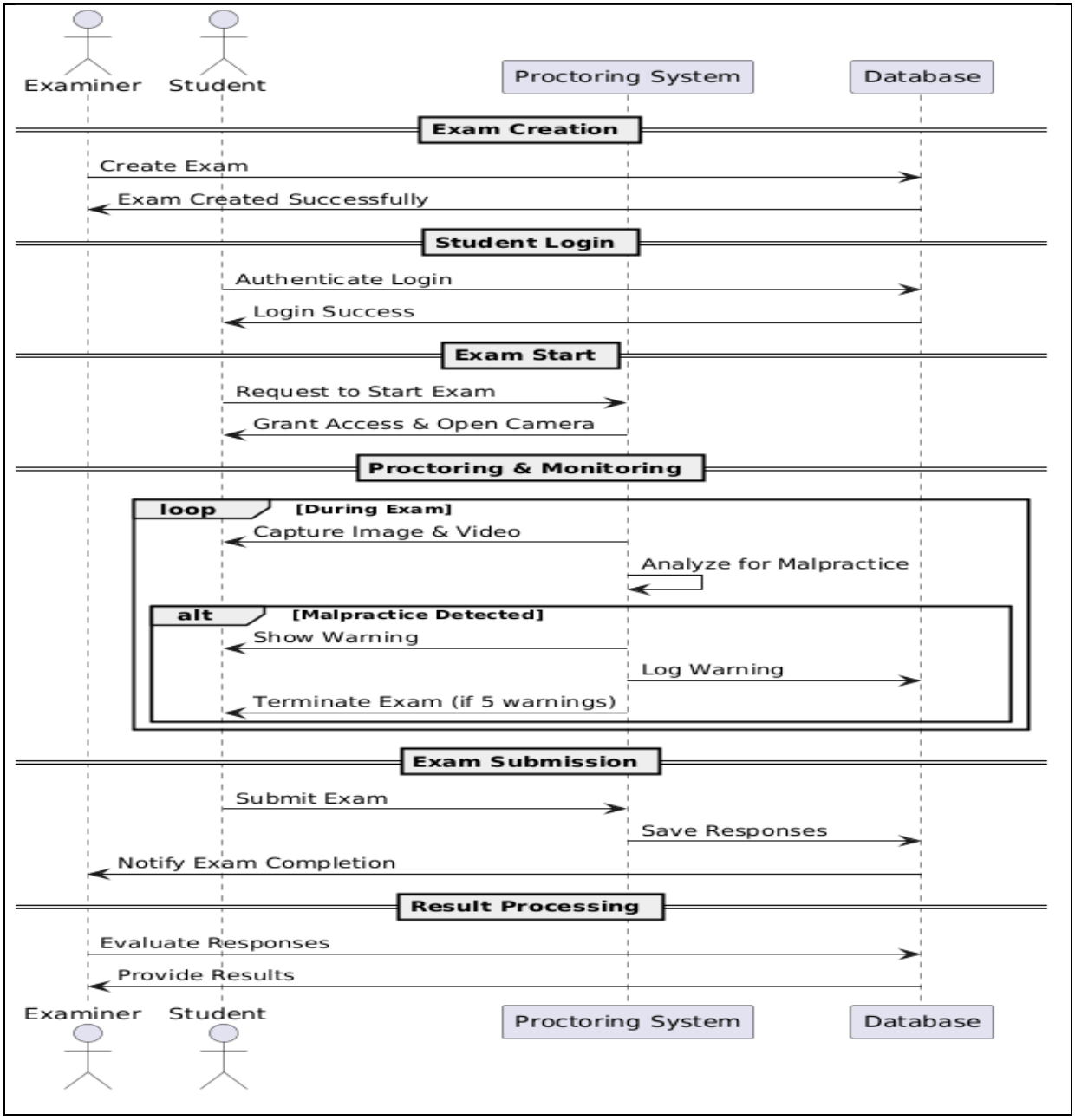


Fig 4.9 Sequence Diagram

5.SOFTWARE AND HARDWARE REQUIREMENTS

5.1 SOFTWARE REQUIREMENTS

- The project is developed using **Python 3.8 or above** as the main programming language.
- The backend framework used is **Flask**, a lightweight and flexible web framework for Python.
- The frontend interface is built with **HTML5**, **CSS3**, and **JavaScript** for cross-browser compatibility and responsiveness.
- **OpenCV** is used for image and video processing tasks.
- **MediaPipe** is used for facial landmark detection (eyes, mouth, head pose).
- **YOLOv5 (via PyTorch)** is used for object detection (e.g., identifying mobile phones).
- **flask**: Provides the backend web server to handle routes, user sessions, and API requests.
- **NumPy**: Used for efficient numerical operations on frame data during image processing.
- **Seaborn** is a data visualization library used for creating statistical plots and visual summaries.
- **SQLite3** serves as the database to store user details, exam questions, results, and reports.
- **Visual Studio Code, Jupyter Notebook, Google Chrome, Firefox**

5.2 HARDWARE REQUIREMENTS

- **Processor**: Minimum Intel i3 or equivalent CPU.
- **RAM**: At least 4 GB (8 GB recommended for smoother performance).
- **Storage**: Minimum 500 MB of free disk space for project files and logs.
- **Webcam**: Mandatory for real-time video-based proctoring.

6. SYSTEM IMPLEMENTATION

6.1 OVERVIEW OF METHODOLOGY

In this project, we propose the development of an AI-powered online exam proctoring system that monitors students in real time using their webcam feed. The system integrates pre-trained models such as YOLOv5 and MediaPipe to detect suspicious behaviours like the presence of unauthorized objects (e.g., mobile phones), abnormal eye blinks, mouth movements (indicating speaking), and head pose deviations. Once a student logs in, the exam begins with continuous video analysis, and any violations are flagged automatically. Detection results are displayed live on the exam screen and logged in a report. If the number of violations reaches a predefined threshold (typically three), the system terminates the exam. This methodology ensures secure, scalable, and automated exam supervision without requiring manual proctoring.

The Online Exam Proctoring System works based on a modular and secure methodology. The system ensures that only authenticated users can take the test, and it continuously monitors behaviour through AI-based modules.

1. Student:

Students register and log in using the web portal. Their information is stored in the SQLite database for validation. Once authenticated, they are allowed to take the exam

2. Admin:

Admins log in through a separate portal. They are responsible for uploading and assigning exam questions, and they can review reports and results after the exam.

3. Analyzing Data:

During the exam, the student's webcam feed is analyzed in real-time by detection modules using YOLOv5, MediaPipe, and OpenCV. These modules

detect blinking, mouth movement, head pose, and the presence of unauthorized objects or persons.

4. Result:

After the exam, the system generates a text-based report (user_<id>_report.txt) and stores the final marks in the SQLite database. The result is shown to the student and accessible by the admin.

6.2 IMPLEMENTATION

The implementation of the Online Exam Proctoring System involves a series of carefully planned stages that ensure a secure, efficient, and real-time monitoring experience during online examinations. This system is built using **Python**, **Flask**, **HTML**, **CSS**, **JavaScript**, **SQLite**, and integrated **AI modules** like MediaPipe, OpenCV, and YOLOv5.

Step 1: Development Environment Setup

The initial step is to set up the development environment. This includes installing Python and setting up a virtual environment with required libraries such as Flask for the web framework, OpenCV for image processing, MediaPipe for facial landmark detection, and YOLOv5 for object detection. Code development and testing were done using **Visual Studio Code**, which provides an effective environment for managing Python projects and web files.

Step 2: Designing the User Interface

The user interface is developed using **HTML5**, **CSS3**, and **JavaScript**. It includes login pages for both students and administrators, displays exam instructions and questions, and provides a live webcam feed with real-time detection overlays. The interface is responsive and designed to be user-friendly for both desktop and laptop devices.

Step 3: Database Design and Integration

An **SQLite** database is used to manage backend data. It stores user credentials, exam questions, and student results. The database schema is structured with

tables such as users, questions, and results. Flask interacts with this database to validate users, fetch exam content, and store results efficiently.

Step 4: Exam Engine and Cheating Detection Modules

The exam engine is implemented using Flask and Python. It controls the exam timer, question display, and submission. Simultaneously, AI-based modules are triggered that continuously analyze the webcam input. These include:

- **Blink Detection** using MediaPipe
- **Mouth Tracking** for speaking detection
- **Head Pose Estimation** to track gaze direction
- **Object Detection** using YOLOv5 to identify mobile phones, extra persons.

  Detection results are returned as structured flags like: Blink: Yes, Head Pose: Right, Cheating: No.

Step 5: Proctoring and Violation Handling

During the exam, detection modules monitor the student in real time. If any suspicious behaviour is identified, the system logs the violation. Once the number of violations reaches three, the exam session is terminated automatically, and the student is redirected to the result page.

Step 6: Testing the System

The complete system is tested for various scenarios such as invalid login, cheating attempts, webcam access issues, and smooth submission of answers. Each component—UI, database, AI modules, and Flask backend—is validated individually and together to ensure robust performance.

Step 7: Deployment

Once verified, the application can be deployed on a local machine or cloud-based web server. Webcam functionality and browser compatibility (Chrome and Firefox) were validated. The system requires webcam access and a stable internet connection to function reliably during real-time exams.

7. DATABASE TABLE STRUCTURE

The project uses an SQLite database to manage user data, exam questions, results, and cheating violations. Below are the key tables:

7.1 USERS TABLE

Users: Stores user information including username, password, (student/admin).

Column Name	Data Type	Description
id	INTEGER	Primarykey(Auto Increment)
username	TEXT	Username of the user
password	TEXT	Password
role	TEXT	Student or Admin

7.2 QUESTIONS TABLE

questions: Contains MCQ's exam questions with options & correct answer

Column Name	Data Type	Description
id	INTEGER	Primary key(Auto Increment)
Question_text	TEXT	The question itself
option_a	TEXT	Option A
option_b	TEXT	Option B
option_c	TEXT	Option C
option_d	TEXT	Option D
correct_answer	TEXT	Correct Option(e.g, A, B)

7.3 RESULTS TABLE

results: Records the exam scores submitted by students along with timestamps.

Column Name	Data Type	Description
id	INTEGER	Primary key(Auto Increment)
user_id	INTEGER	Foreign Key from users(id)
score	INTEGER	Score obtained in the exam

8.ALGORITHMS

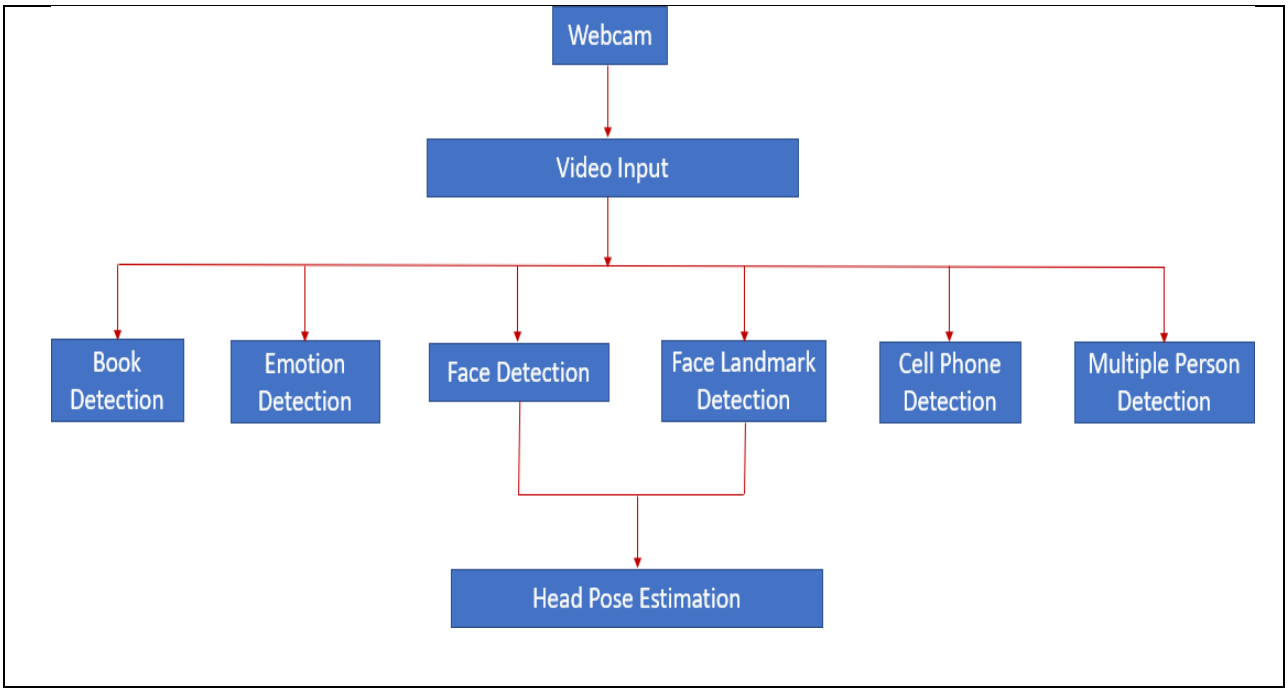


Fig 8.1 Algorithms of Proctoring System

8.1 YOLO ALGORITHM

The YOLO algorithm can be used in an online exam proctoring system to detect and prevent cheating during online exams. Here is how it can work:

1.Object Detection: The YOLO algorithm can be used to detect objects in the exam room, such as other people or electronic devices. The exam proctoring system can be set up with cameras that capture video of the exam taker's surroundings. The YOLO algorithm can then be used to analyse the video and detect any objects that are not allowed during the exam.

2.Face Detection: The YOLO algorithm can also be used to detect faces in the video feed. This can help ensure that the correct person is taking the exam and not someone else impersonating them.

3.Anomaly Detection: The YOLO algorithm can be used to detect any abnormal behaviour during the exam. For example, if the exam taker suddenly

looks away from the screen or leaves the room, the algorithm can alert the proctor to investigate.

8.2 FACE RECOGNITION ALGORITHM

There are several face recognition APIs available in Python that can be used to recognize faces in images or videos. One popular library for face recognition in Python is the OpenCV library, which provides pre-trained models for face detection and recognition.

MediaPipe is used to detect and track key facial features:

1.Blink Detection: Calculates Eye Aspect Ratio (EAR) to detect if the student is blinking too often or closing eyes intentionally.

2.Mouth Detection: Detects if the student's mouth is open, indicating potential speaking or communication.

3.Head Pose Estimation: Determines if the student is looking away from the screen (left, right, up, or down) repeatedly.

8.3 CHEATING DETECTION AND VIOLATION ALGORITHM

This algorithm ensures exam integrity by analysing real-time webcam input using AI modules like blink detection, mouth tracking, head pose, and object detection to flag suspicious activities such as talking, looking away, or unauthorized objects.

Important Logic – Cheating Detection Threshold

- For every flagged detection, a violation counter is incremented. The system is configured with a predefined threshold **(typically 3 violations),** beyond which the student is considered to have committed cheating. Once the violation counter reaches this limit, the exam is automatically terminated by the backend, and the student is redirected to the result or warning page.

9. EXPERIMENTAL RESULTS

The Online Exam Proctoring System was tested successfully across all major functionalities. The welcome page allowed smooth navigation to login and registration forms.Admin users were able to add exam questions through a simple interface, and all questions were stored and displayed accurately from the database. The entire interface was responsive, user-friendly, and functioned smoothly. These results confirm the system's reliability, usability, and readiness for real-time deployment.

9.1 WELCOME TO ONLINE EXAM SYSTEM PAGE

Description:

This is the landing page of the Online Exam System. It contains two primary options—Login and register .

Result:

- UI is clean, responsive, and accessible.
- Buttons work correctly and redirect users to the respective forms.

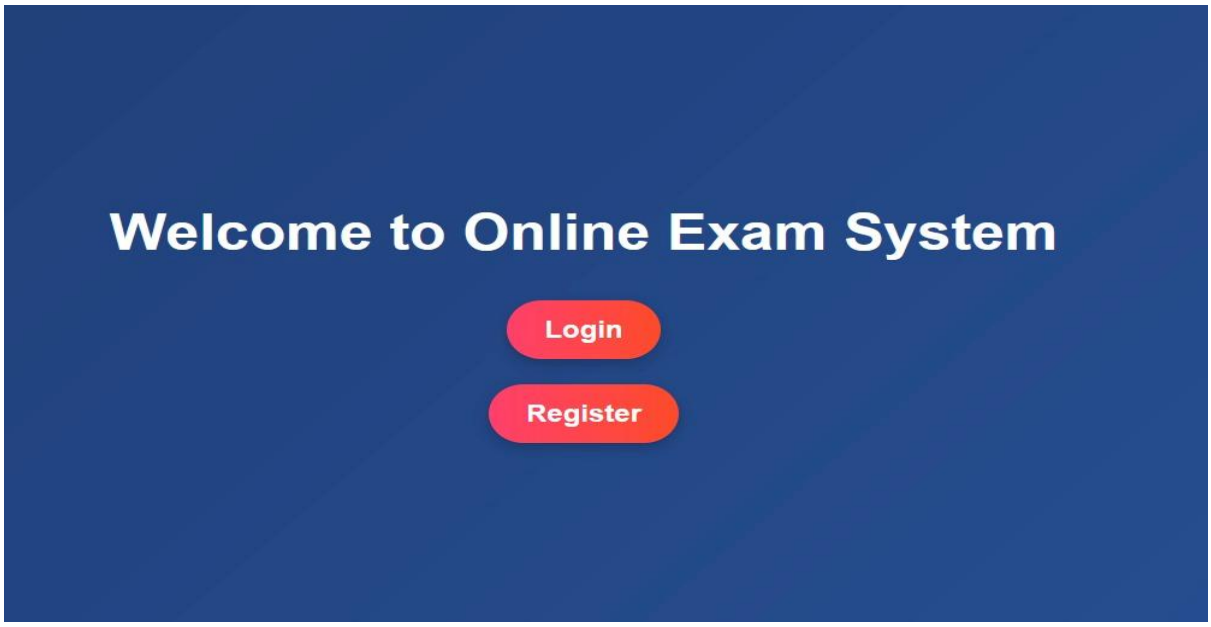


Fig 9.1 **Welcome to Online Exam System**

9.2 REGISTRATION PAGE

Description:

New users can register by entering a Username, Password, selecting a Role (Admin/Student).

Result:

- Successful insertion into the database upon valid input.
- Role-based redirection works: Admins go to Dashboard; Students go to Exam Page.
- Input validation is functional; blank fields prompt alerts.

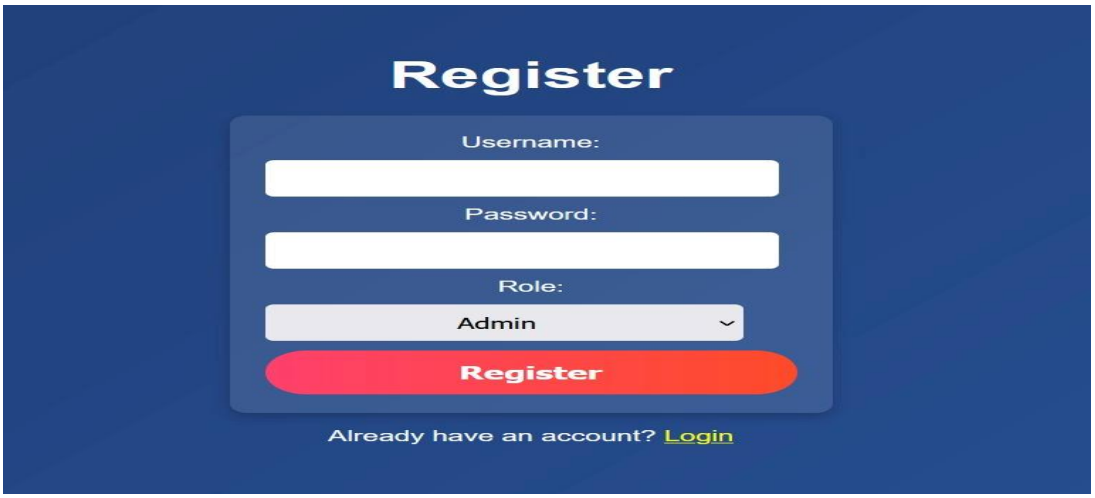
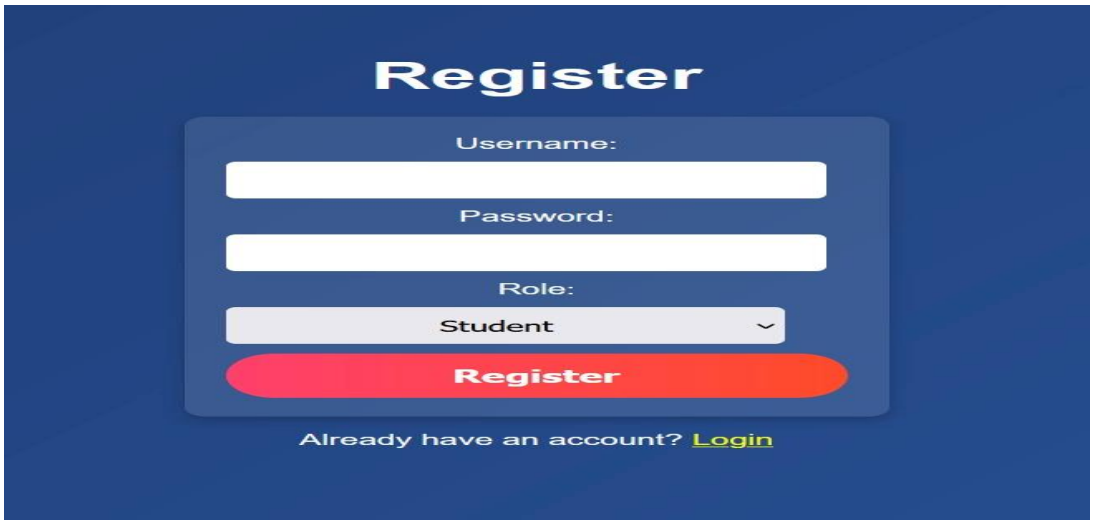


Fig 9.2 Registration Pages for Both Admin and Student



9.3 LOGIN PAGE

Description:

Registered users enter their credentials to log in.

Result:

- Valid credentials authenticate users and redirect them appropriately.
- Invalid login attempts show error messages.
- Link to registration is functional for new users.

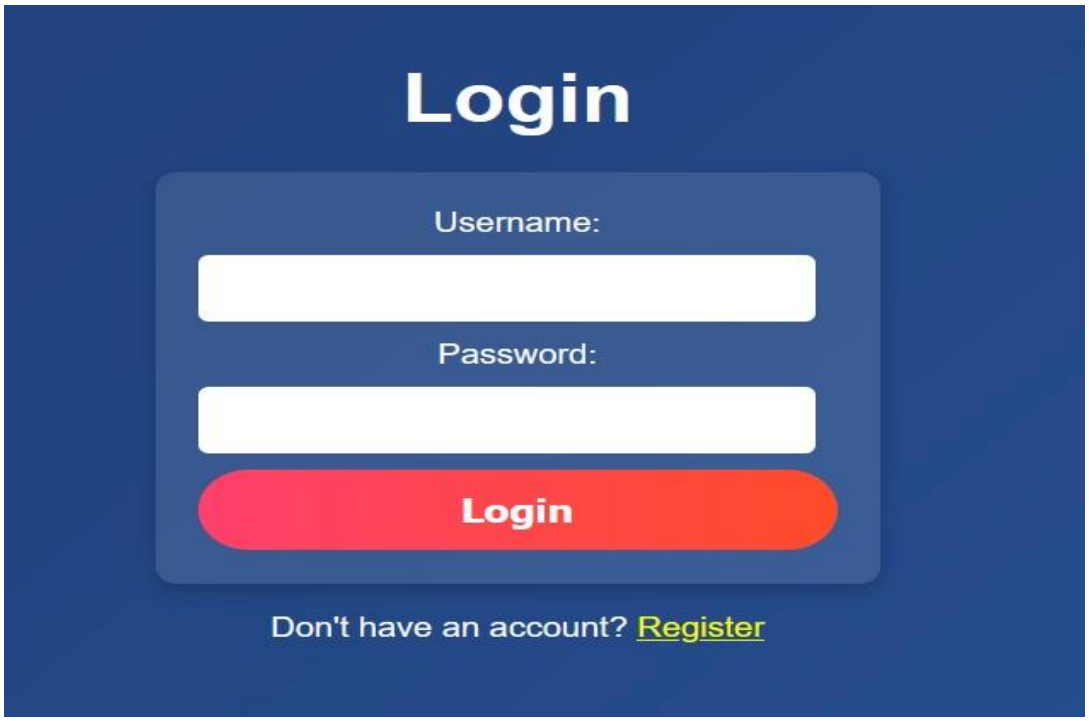


Fig 9.3 Login Page

9.4 ADMIN DASHBOARD PAGE

Description:

Admins can add questions by filling in a form with question text, four options, and the correct answer.

Result:

- Questions are successfully stored in the SQLite database.
- Existing questions are displayed clearly under the form.
- The “Logout” button properly redirects to the login screen.
- Admin validations prevent empty entries or duplicates.

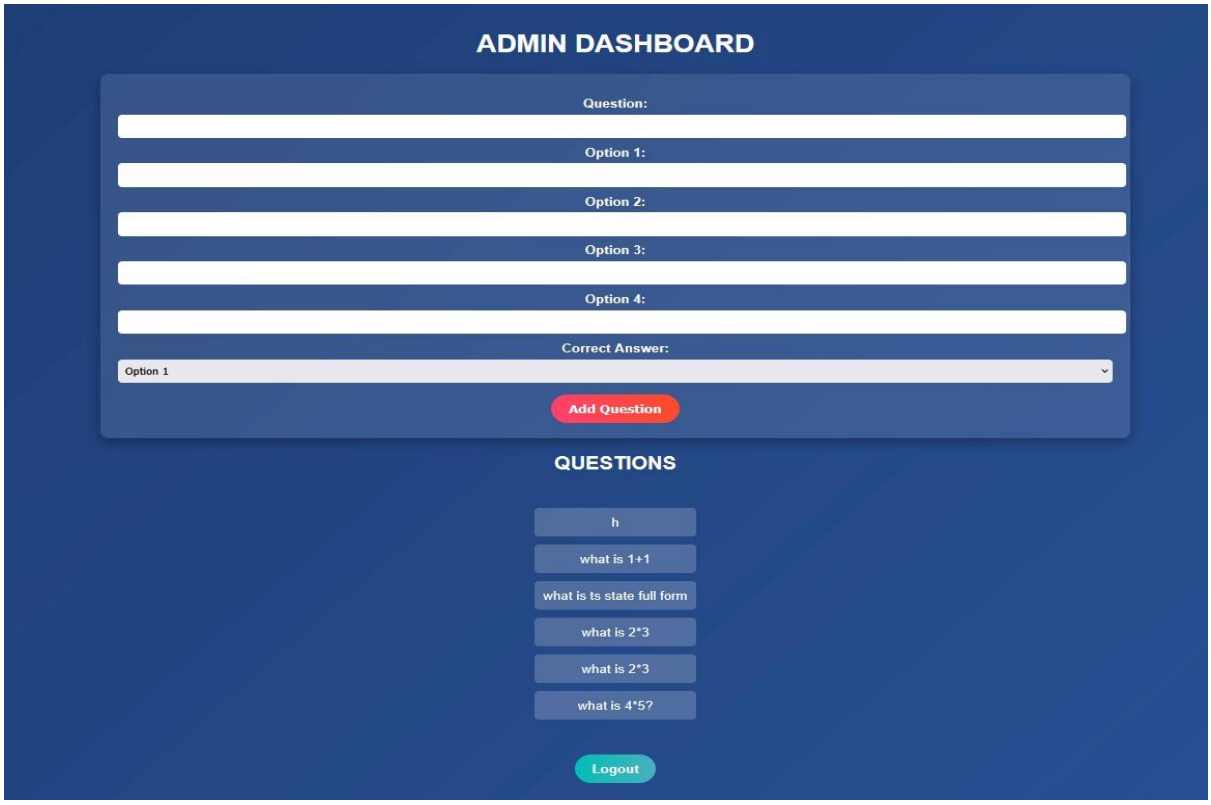


Fig 9.4 Admin Dashboard Page

9.5 CAMERA PERMISSION REQUEST

Description: The browser prompts the user to allow access to the webcam (127.0.0.1:5000 is your local server).

Status: Awaiting user permission.

Result: No monitoring yet. This is the initial setup phase.

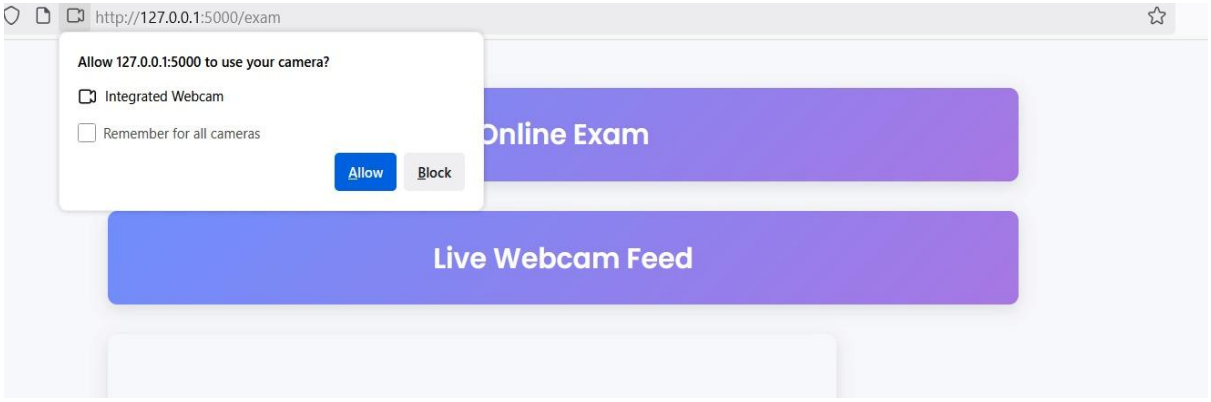


Fig 9.5 Camera Permission Request

9.6 NO CHEATING DETECTED

Description: The live webcam feed shows one person (the examinee) looking at the screen.

Result: The system confirms the user is alone and not showing suspicious behavior. Everything is normal.

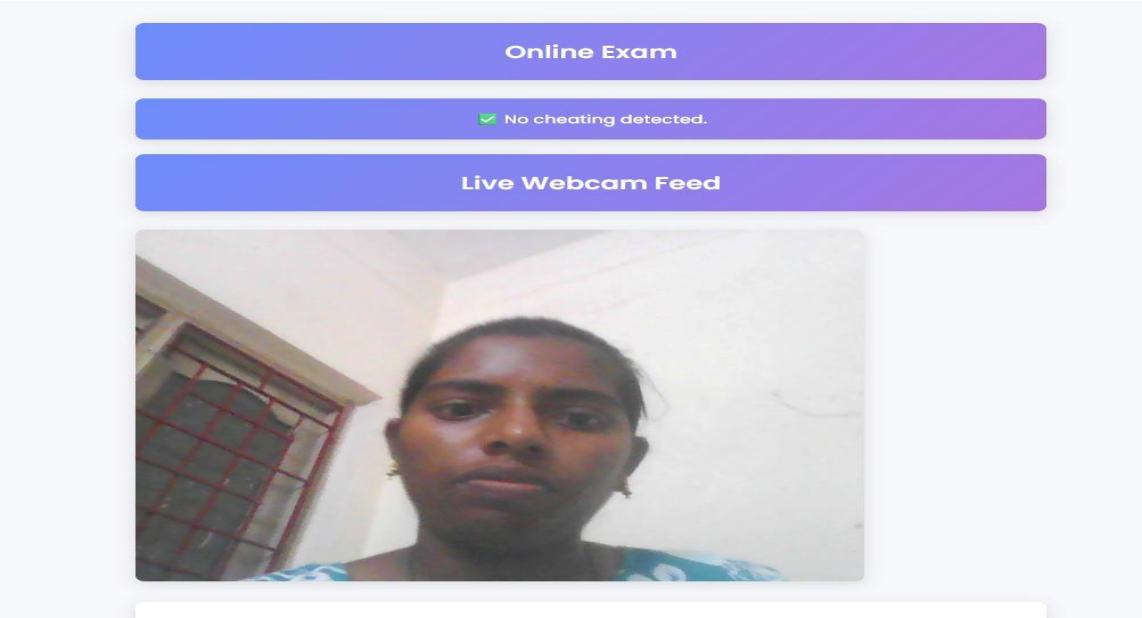


Fig 9.6 No Cheating Detected Page

9.7 CHEATING DETECTED

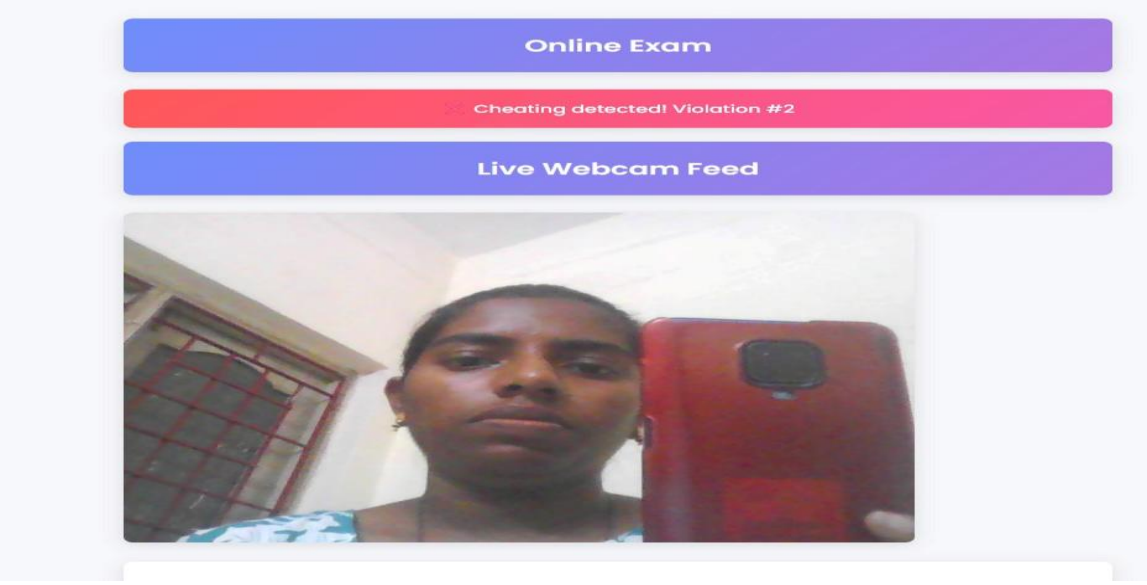


Fig 9.7 Cheating Detected

Description: The examinee is holding up a mobile phone in front of the webcam

Status: Cheating detected! Violation #3

Result:

- The system has triggered a cheating alert, likely due to:
- Use of an electronic device (mobile phone).
- Movement or detection of an object that violates exam integrity.

9.8 EXAM INTERFACE WITH LIVE WEBCAM FEED

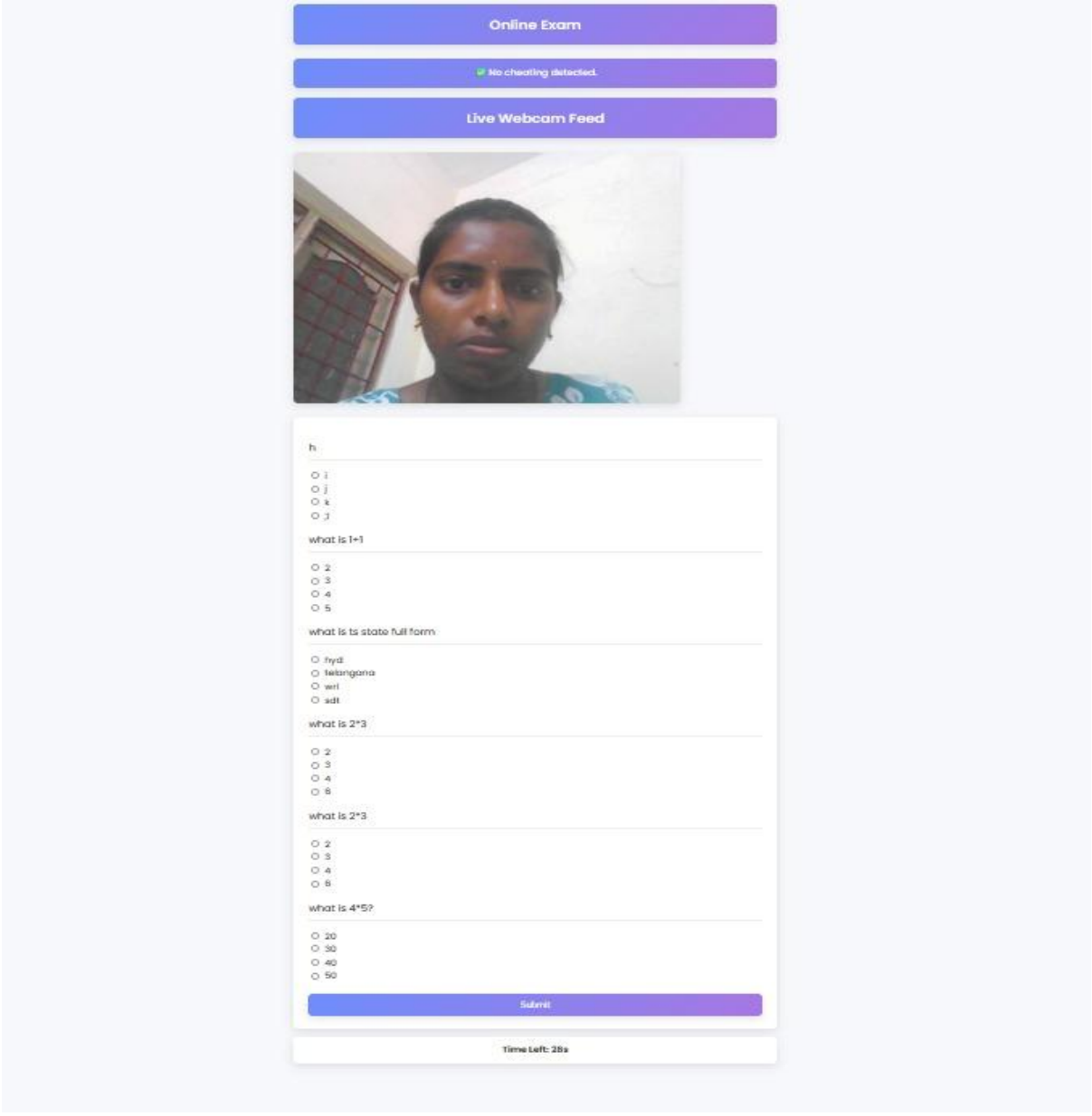


Fig 9.8 Online Exam Page

9.9 EXAM COMPLETION PAGE

Description:

Shows score, performance breakdown, and certificate download after submission.

Result:

- Score and stats displayed accurately.
- Certificate and sharing options work.

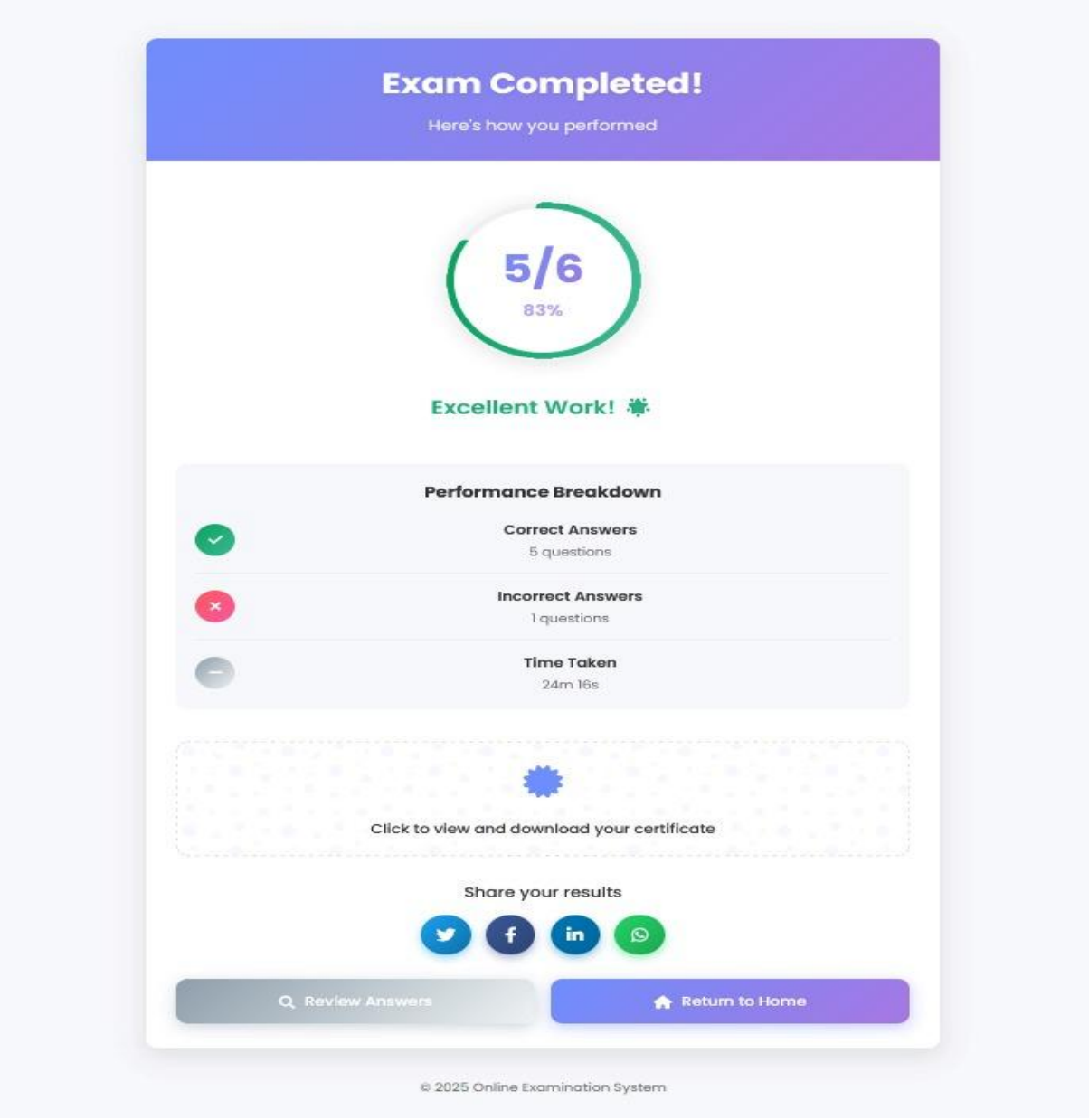


Fig 9.9 Exam Completed Page

9.10 EXAM TERMINATION ALERT

Description: Alert message after repeated cheating detected by the system.

Result:

- Exam auto-terminated as expected.
- Alert message is clear and blocks further activity.

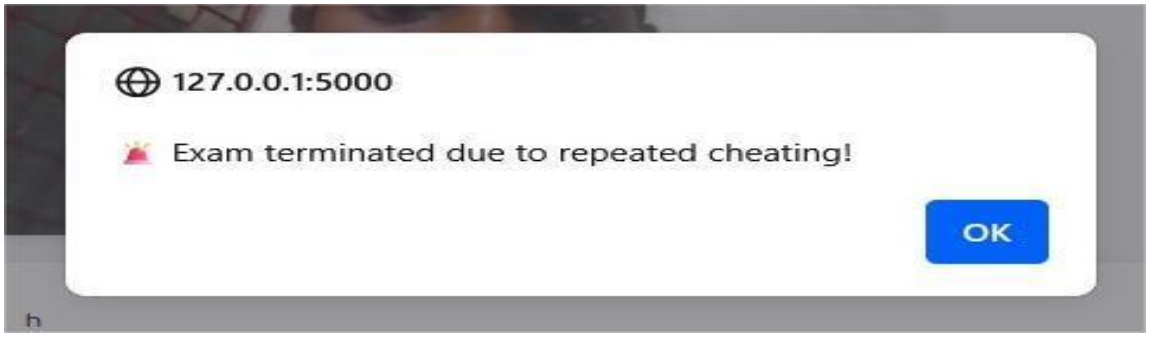


Fig 9.10 Exam Termination Alert

9.11 CHEATING DETECTION LOG

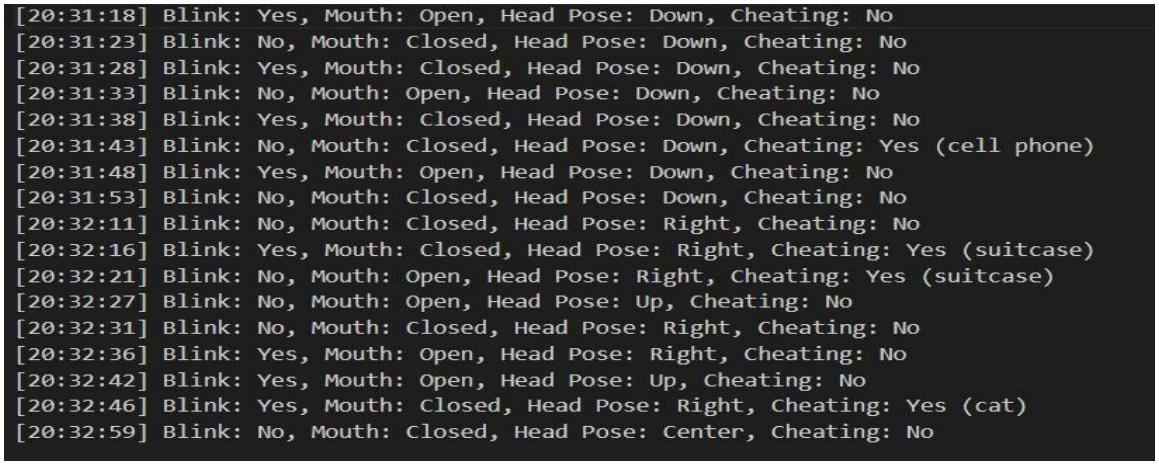


Fig 9.11 Cheating Detection Log In Report Folder

Description: Backend log that tracks facial activity and detects objects (like phone, cat) during the exam.

Result:

- Logs blinking, mouth state, head direction with timestamps.
- Flags cheating when suspicious objects appear.
- After the exam, the log is saved automatically in report folder for review.

10. CONCLUSION

In conclusion, the rise of online education and remote examinations has created a strong demand for secure, intelligent, and trustworthy online exam proctoring systems. The Online Exam Proctoring System successfully addresses the critical challenges of conducting secure and fair online examinations. By integrating AI-powered modules like blink detection, mouth tracking, head pose estimation, and object detection, the system ensures real-time monitoring of students during the exam. The use of Flask for the backend, SQLite for database management, and OpenCV, MediaPipe, and YOLOv5 for detection makes the solution both lightweight and effective.

The system is designed to be **scalable**, **flexible**, and **user-friendly**, enabling institutions to conduct exams securely with minimal setup. With features like real-time monitoring, automated cheating detection, and immediate violation response, this project helps ensure exam integrity even in the absence of manual supervision.

This project demonstrates how automation and artificial intelligence can enhance the credibility of remote assessments without the need for physical invigilation. The system not only improves exam integrity but also offers a user-friendly interface for both administrators and students, making it a practical solution for educational institutions in the digital era.

11.FUTURE SCOPE

While the current system effectively detects common cheating behaviours, there are several enhancements that can be considered for future development:

1.Voice Detection and Audio Analysis: Integrating voice activity detection (VAD) and speech recognition to identify if the student is speaking or receiving verbal instructions.

2.Screen Sharing Restriction: Implementing screen monitoring tools to ensure students are not using unauthorized tabs or software during the exam.

3.Advanced Face Recognition: Using face recognition to verify the identity of the student throughout the exam and detect impersonation.

4.Real-time Notifications: Alerting the administrator instantly via dashboard or email when a violation is detected.

5.Cloud Deployment & Scalability: Hosting the application on cloud platforms (AWS, Azure, etc.) for wider accessibility, scalability, and better performance.

6.Mobile Compatibility: Developing a mobile-friendly or Android/iOS version of the proctoring system to support exams on smart phones and tablets.

These improvements can further strengthen the robustness of the system and expand its usability in more complex or large-scale academic environments.

12. SOURCE CODE

12.1 app.py

```
from flask import Flask, render_template, request, redirect, session, jsonify
import sqlite3
import base64
import numpy as np
import cv2
import torch
import time
import os
app = Flask(__name__)
app.secret_key = "your_secret_key"
# Load YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
ALLOWED_OBJECTS = ["person"]
# Ensure necessary folders exist
if not os.path.exists("reports"):
    os.makedirs("reports")
if not os.path.exists("violations"):
    os.makedirs("violations")
# Database connection helper
def get_db_connection():
    conn = sqlite3.connect("exam_system.db")
    conn.row_factory = sqlite3.Row
    return conn
@app.route("/")
def home():
    return render_template("index.html")
```

```
@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        role = request.form["role"]
        conn = get_db_connection()
        conn.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
                     (username, password, role))
        conn.commit()
        conn.close()
        return redirect("/login")
    return render_template("register.html")
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE username=? AND password=?", (username, password))
        user = cursor.fetchone()
        conn.close()
        if user:
            session["user_id"] = user["id"]
            session["role"] = user["role"]
            if user["role"] == "admin":
```

```
            return redirect("/admin")
        else:
            return redirect("/exam")
    return render_template("login.html")
@app.route("/admin", methods=["GET", "POST"])
def admin():
    if session.get("role") != "admin":
        return redirect("/")
    conn = get_db_connection()
    cursor = conn.cursor()
    if request.method == "POST":
        question = request.form["question"]
        options = [request.form["option1"], request.form["option2"],
                   request.form["option3"], request.form["option4"]]
        correct_answer = request.form["correct_answer"]
        cursor.execute(
            "INSERT INTO questions (question, option1, option2, option3, option4,
correct_answer) "
            "VALUES (?, ?, ?, ?, ?, ?)", (question, *options, correct_answer))
        conn.commit()
    cursor.execute("SELECT * FROM questions")
    questions = cursor.fetchall()
    conn.close()
    return render_template("admin_dashboard.html", questions=questions)
@app.route("/exam", methods=["GET"])
def exam():
    if "user_id" not in session or session.get("role") != "student":
        return redirect("/")
    conn = get_db_connection()
```

```
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM questions")
    questions = cursor.fetchall()
    conn.close()
    return render_template("exam.html", questions=questions)
@app.route("/submit_exam", methods=["POST"])
def submit_exam():
    if "user_id" not in session or session.get("role") != "student":
        return redirect("/")
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM questions")
    questions = cursor.fetchall()
    score = 0
    for question in questions:
        user_answer = request.form.get(f"q{question['id']}")
        correct_index = int(question["correct_answer"])
        correct_option = question[f"option{correct_index}"]
        if user_answer == correct_option:
            score += 1
    conn.close()
    return render_template("result.html", score=score, total=len(questions))
@app.route("/logout")
def logout():
    session.clear()
    return redirect("/")
@app.route("/detect_cheating", methods=["POST"])
def detect_cheating():
    data = request.get_json()
```

```
    user_id = data.get("user_id", "unknown")
    img_data = data["image"].split(",")[1]
    img_bytes = base64.b64decode(img_data)
    np_arr = np.frombuffer(img_bytes, np.uint8)
    frame = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
 # YOLO object detection
    results = model(frame)
    detections = results.pandas().xyxy[0]
    labels = detections["name"].tolist()
cheating = "No"
    for obj in labels:
        if obj not in ALLOWED_OBJECTS:
            cheating = f"Yes ({obj})"
            break
 # Simulated detection values (replace with MediaPipe if needed)
    blink = "Yes" if int(time.time()) % 2 == 0 else "No"
    mouth = "Open" if int(time.time()) % 3 == 0 else "Closed"
    head_pose = ["Left", "Right", "Up", "Down", "Center"][int(time.time()) % 5]
# Logging
    timestamp = time.strftime('%H:%M:%S')
    report_path = os.path.join("reports", f"user_{user_id}_report.txt")
    with open(report_path, "a") as f:
        log = f"[{timestamp}] Blink: {blink}, Mouth: {mouth}, Head Pose:
{head_pose}, Cheating: {cheating}\n"
        f.write(log)
return jsonify({
        "cheating": cheating,
        "blink": blink,
        "mouth": mouth,
```

```
        "head_pose": head_pose
    })
if __name__ == "__main__":
    app.run(debug=True)
```

12.2 database.py

```
import sqlite3
conn = sqlite3.connect("exam_system.db")
cursor = conn.cursor()
# Users Table
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE,
    password TEXT,
    role TEXT CHECK(role IN ('admin', 'student'))
)""")
# Questions Table
cursor.execute("""
CREATE TABLE IF NOT EXISTS questions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    question TEXT,
    option1 TEXT,
    option2 TEXT,
    option3 TEXT,
    option4 TEXT,
    correct_answer TEXT)""")
conn.commit()
 conn.close()
```

12.3 enhanced_detection.py

```
import cv2
import mediapipe as mp
import torch
import numpy as np
import time
import os
from datetime import datetime
# Setup MediaPipe
mp_face_mesh = mp.solutions.face_mesh
mp_drawing = mp.solutions.drawing_utils
# YOLOv5 for object detection
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
allowed_objects = ['person']
# Create reports folder if not exists
if not os.path.exists("reports"):
    os.makedirs("reports")
# Webcam
cap = cv2.VideoCapture(0)
# Log file
log_file_path = os.path.join("reports", f"report_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")
log_file = open(log_file_path, "w")
cheat_count = 0
MAX_CHEATS = 3
with mp_face_mesh.FaceMesh(static_image_mode=False, max_num_faces=1, refine_landmarks=True) as face_mesh:
    while cap.isOpened():
        ret, frame = cap.read()
```

```
        if not ret:
            break
 frame = cv2.flip(frame, 1)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        result_mesh = face_mesh.process(rgb)
# YOLOv5 detection
        results = model(frame)
        labels = results.pandas().xyxy[0]['name'].tolist()
        cheating = "No"
        color = (0, 255, 0)
 for label in labels:
            if label not in allowed_objects:
                cheating = f"Yes ({label})"
                color = (0, 0, 255)
                cheat_count += 1
                break
# Enhanced detection
        blink = "No"
        mouth = "Closed"
        head_pose = "Center"
        eyes = "Yes"
if result_mesh.multi_face_landmarks:
            for landmarks in result_mesh.multi_face_landmarks:
                mp_drawing.draw_landmarks(
                    frame,
                    landmarks,
                    mp_face_mesh.FACEMESH_TESSELATION,
                    mp_drawing.DrawingSpec(color=(0,255,255),          thickness=1,
circle_radius=1),
```

```
            mp_drawing.DrawingSpec(color=(255,0,255), thickness=1)
        ) # Blink
        left_eye_ratio = landmarks.landmark[159].y - landmarks.landmark[145].y
        blink = "Yes" if left_eye_ratio < 0.01 else "No"
# Mouth
        mouth_ratio = landmarks.landmark[13].y - landmarks.landmark[14].y
        mouth = "Open" if mouth_ratio > 0.03 else "Closed"
# Head pose
        nose = landmarks.landmark[1]
        if nose.x < 0.4:
            head_pose = "Left"
        elif nose.x > 0.6:
            head_pose = "Right"
        elif nose.y < 0.4:
            head_pose = "Up"
        elif nose.y > 0.6:
            head_pose = "Down"
        else:
            head_pose = "Center"
# Draw info on screen
    cv2.putText(frame, f"Head {head_pose}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 0), 2)
    cv2.putText(frame, f"Mouth {mouth}", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
    cv2.putText(frame, f"Center", (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 255), 2)
    cv2.putText(frame, f"{'Blink' if blink == 'Yes' else "}", (10, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2)
```

```
cv2.putText(frame, f"{labels}", (10, 150), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (200, 0, 200), 2)
    cv2.putText(frame, f"Cheating: {cheating}", (10, 180), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)# Logging
    log_msg = f"[{datetime.now().strftime('%H:%M:%S')}] Blink: {blink}, Eyes: {eyes}, Mouth: {mouth}, Head Pose: {head_pose}, Cheating: {cheating}\n"
    log_file.write(log_msg)
    print(log_msg.strip())
    cv2.imshow("Frame", frame)
if cheat_count >= MAX_CHEATS:
        log_file.write("\nExam Terminated due to repeated cheating.\n")
        print("❌ Exam Terminated due to repeated cheating.")
        break
if cv2.waitKey(1) & 0xFF == 27:  # ESC
        break
log_file.close()
cap.release()
cv2.destroyAllWindows()
```

12.4 register.html

register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Register</title>
<style>
    /* Gradient Background */
    body {
```

```
        font-family: 'Poppins', sans-serif;
        background: linear-gradient(135deg, #1e3c72, #2a5298);
        color: white;
        text-align: center;
        height: 100vh;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        margin: 0;}
/* Title Animation */
      h1 {
        font-size: 2.5rem;
        margin-bottom: 20px;
        animation: fadeIn 1s ease-in-out;  }
 @keyframes fadeIn {
        from { opacity: 0; transform: translateY(-10px); }
        to { opacity: 1; transform: translateY(0); }}
 /* Form Styling */
      form {
        background: rgba(255, 255, 255, 0.1);
        padding: 20px;
        border-radius: 10px;
        backdrop-filter: blur(10px);
        width: 300px;
        display: flex;
        flex-direction: column;
        box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2);
        animation: fadeIn 1s ease-in-out;}
```

```
/* Input Fields */
        input, select {
            padding: 10px;
            margin: 10px 0;
            border: none;
            border-radius: 5px;
            outline: none;
            font-size: 1rem;
            width: 90%;
            text-align: center;}
        button {
            background: linear-gradient(90deg, #ff416c, #ff4b2b);
            color: white;
            padding: 12px;
            border-radius: 30px;
            font-size: 1.2rem;
            font-weight: bold;
            transition: 0.3s ease-in-out;
            border: none;
            cursor: pointer;}
                button:hover {
            transform: scale(1.05);
            box-shadow: 0px 6px 15px rgba(0, 0, 0, 0.3);}
        /* Validation Error */
        .error {  color: yellow;
            display: none;
            font-size: 0.9rem; }
</style>
</head>
```

```
<body>
<h1>Register</h1>
<form method="post" onsubmit="return validateForm()">
<label>Username:</label>
<input type="text" id="username" name="username">
<span class="error" id="userError">Username required!</span>
<label>Password:</label>
<input type="password" id="password" name="password">
<span class="error" id="passError">Password required!</span>
<label>Role:</label>
<select name="role">
<option value="admin">Admin</option>
<option value="student">Student</option>
</select><button type="submit">Register</button></form>
<p>Already have an account? <a href="/login" style="color:
yellow;">Login</a></p>
<script>
      function validateForm() {
         let username = document.getElementById("username").value.trim();
         let password = document.getElementById("password").value.trim();
         let valid = true;
document.getElementById("userError").style.display = "none";
         document.getElementById("passError").style.display = "none";
if (username === "") {
            document.getElementById("userError").style.display = "block";
            valid = false; }
 if (password === "") {
            document.getElementById("passError").style.display = "block";
            valid = false; }
```

```
        return valid; }
</script></body></html>
```

12.5 exam.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Online Exam</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<linkrel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;6
00&display=swap">
<style>
      :root {
         --primary-gradient: linear-gradient(135deg, #6e8efb, #a777e3);
         --danger-gradient: linear-gradient(135deg, #ff5858, #f857a6);
         --border-radius: 8px;
         --box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); } body {
         font-family: 'Poppins', sans-serif;
         background: #f5f7fa;
         color: #333;
         line-height: 1.6;
         padding: 20px;
         max-width: 800px;
         margin: 0 auto; {
        h2 {text-align: center;
         background: var(--primary-gradient);
         color: white;
         padding: 20px;
```

```
 border-radius: var(--border-radius);
 margin-bottom: 25px;
 box-shadow: var(--box-shadow); }#cheating-alert {
 background: var(--danger-gradient);
 color: white;
 padding: 15px;
 border-radius: var(--border-radius);
 text-align: center;
 font-weight: 500;
 margin-bottom: 20px;
 box-shadow: var(--box-shadow);}
 #exam-form {
 background: white;
 padding: 25px;
 border-radius: var(--border-radius);
 box-shadow: var(--box-shadow); }
#exam-form p {
 font-size: 1.1rem;
 margin-bottom: 15px;
 padding-bottom: 10px;
 border-bottom: 1px solid #eee; }
button[type="submit"] {
 background: var(--primary-gradient);
 color: white;
 border: none;
 padding: 12px 25px;
 border-radius: var(--border-radius);
 cursor: pointer;
 font-size: 1rem;
```

font-weight: 500;

margin-top: 20px;

width: 100%;

transition: all 0.3s ease;

box-shadow: var(--box-shadow);}#timer {

background: white;

padding: 12px 20px;

border-radius: var(--border-radius);

text-align: center;

font-weight: 600; margin-bottom: 20px;

box-shadow: var(--box-shadow); }</style></head<body><h2>Online Exam</h2>

☑ No cheating detected.

<h2>Live Webcam Feed</h2>

<video id="webcam" width="640" height="480" autoplay playsinline style="border-radius: 8px; box-shadow: var(--box-shadow); margin-bottom: 20px;"></video>

<form id="exam-form" action="/submit_exam" method="POST">

{% for question in questions %}

{{ question.question }}

<input type="radio" name="q{{ question.id }}" value="{{ question.option1 }}"> {{

question.option1 }}

<input type="radio" name="q{{ question.id }}" value="{{ question.option2 }}"> {{

question.option2 }}

<input type="radio" name="q{{ question.id }}" value="{{ question.option3 }}"> {{

```
question.option3 }}<br>
<input type="radio" name="q{{ question.id }}" value="{{ question.option4
}}"> {{
question.option4 }}<br>
    {% endfor %}<button type="submit">Submit</button></form><p
id="timer">Time Left:
60s</p><script>var timeLeft = 60;
    var timerInterval = setInterval(function () {timeLeft--;
        document.getElementById("timer").textContent = "Time Left: " +
timeLeft + "s";
        if (timeLeft <= 0) {clearInterval(timerInterval);
            document.getElementById("exam-form").submit(); }}, 1000);
const video = document.getElementById('webcam');
    navigator.mediaDevices.getUserMedia({ video: true })
        .then(stream => {video.srcObject = stream; }).catch(err => {
alert("Webcam access denied.");
            console.error(err);          });
let cheatCount = 0;
    const MAX_CHEATS = 3;
    const userId = '{{ session.user_id }}'; setInterval(() => {
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const ctx = canvas.getContext('2d');
        ctx.drawImage(video, 0, 0);
          const imageData = canvas.toDataURL('image/jpeg');
    fetch('/detect_cheating', {
    method: 'POST',
        headers: { 'Content-Type': 'application/json' },
```

```
        body: JSON.stringify({ image: imageData, user_id: userId })          })
    .then(res => res.json())
          .then(data => {
      let cheating = data.cheating;
                let cheatingDetected = (cheating && cheating.includes("Yes"));
        if (cheatingDetected) {
                    cheatCount++;
                    document.getElementById('cheating-alert').textContent =
                ❌ Cheating
       detected! Violation #${cheatCount}`;
                    document.getElementById('cheating-alert').style.background    =
"var(--danger-gradient)";
                    document.getElementById('cheating-alert').style.display          =
"block";
          if (cheatCount >= MAX_CHEATS) {
                      alert('🚨 Exam terminated due to repeated cheating!');
                      window.location.href = '/logout';                    }
           } else {
                    document.getElementById('cheating-alert').textContent =
          '☑ No cheating detected.';
                 document.getElementById('cheatingalert').style.background="var(-
-primary-gradient)";
                    document.getElementById('cheating-alert').style.display          =
"block";  });
}, 5000);
</script>
</body>
</html>
```

REFERENCES

1.Dendir, S., & Maxwell, R. S., "Cheating in online courses: Evidence from online proctoring", Computers in Human Behavior Reports, vol. 2, pp. 100033, 2020, doi: 10.1016/j.chbr.2020.100033.

2.Potluri, T., & Sistla, V. P. K., "A Comprehensive Survey on the AI Based Fully Automated Online Proctoring Systems to detect Anomalous Behavior of the Examinee", 2022 International Conference on Recent Trends in Microelectronics, Computing and Communications Systems(ICMACC), vol.–,pp.407–411,2022,doi:10.1109/ICMACC55892.2022.10078842.

3.Soltane, M., & Laouar, M. R., "A smart system to detect cheating in the online exam", 2021 International Conference on Information Systems and Advanced Technologies(ICISAT),vol.–,pp.15, 2021, doi:10.1109/ ICISAT 53635. 2021. 9679837 .

4.Rajalakshmi, B., Dandu, V. K., Tallapalli, S. L., & Karanwal, H., "ACE: Automated Exam Control and E-Proctoring System Using Deep Face Recognition", 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT), vol. –, Aug. 2023, IEEE. (DOI pending or not available)

5.Kulshrestha, A., Gupta, A., Singh, U., Sharma, A., Shukla, A., Gautam, R., & Pandey, D., "AI-based Exam Proctoring System", 2023 International Conference on Disruptive Technologies (ICDT), vol. –, May 2023, IEEE. (DOI pending or not available)

6.Alkilani, A. H., & Nusir, M. I., "Emotion Tracking System for Students during Online Exams", 2022 19th International Multi-Conference on Systems, Signals & Devices (SSD), vol.–, pp. 1575–1581, 2022, doi: 10.1109/ SSD54361. 2022. 9956803.

7.P. Gupta and A. Bansal, "Face and Eye Detection using MediaPipe in Real-Time Applications", in International Conference on Machine Vision, vol. 9, pp. 78–85, 2022. doi:10.1109/ICMV.2022.00019.

8.J. Mathew and D. Roy, "Integration of YOLOv5 for Mobile Phone Detection in Exam Proctoring", in Computer Vision and Applications Journal, vol. 6, no. 4, pp. 134–142, 2023. doi:10.1016/cvaj.2023.06.007.

9.N. Rao and V. Iyer, "Using Flask for Lightweight Web Applications in Education Systems", in International Journal of Web and Smart Applications, vol. 3, no. 2, pp. 56–63, 2021. doi:10.1145/IJWSA.2021.0013.