# ABSTRACT

In recent years, the integrity of competitive examinations has come under serious threat due to increasing instances of high-tech and organized cheating. Notably, several cases in entrance exams such as the Joint Entrance Examination (JEE) have revealed how candidates manipulated proctoring systems and used unfair means—some even gaining admission into premier institutions under false merit. These incidents not only undermine the spirit of meritocracy but also damage the credibility of national assessment systems, leading to long-term harm for genuine students and, ultimately, the intellectual future of the country.

As India and the world gradually shift towards digital education and online assessments, it is critical to design examination systems that are both scalable and secure. This project presents a prototype of a **Cheating Detection System** that uses Python and its powerful libraries to ensure fair conduct in online exams. By integrating modules such as face recognition, object detection (e.g., phones, books), multi-person detection, and browser monitoring, the system can identify and flag suspicious activity in real time.

This AI-powered system not only acts as a digital invigilator but also makes it feasible to conduct large-scale online exams with minimal human supervision. With further development, such systems can be standardized and adopted by educational boards, colleges, and even national testing agencies to promote transparent, accessible, and corruption-free evaluation. This work lays a foundation for a future where online exams can become more inclusive and widespread—without compromising on fairness or academic honesty.

# TABLE OF CONTENTS

# CHAPTER 1 : INTRODUCTION

Online examinations have become increasingly common due to the rapid digitalization of education and the need for remote learning solutions. However, with this shift comes the growing challenge of ensuring academic integrity in a virtual environment. Traditional invigilation methods are ineffective in online scenarios, and cheating during exams has become easier and more frequent. To address this problem, our project aims to build a **Cheating Detection System** using **Python and its libraries** to monitor and control dishonest behavior during online examinations.

Our system integrates **face recognition**, **object detection**, **eye/gaze tracking**, and **multiple person detection** using real-time webcam input. It actively monitors the candidate throughout the exam, detects any suspicious activity, and issues **automated alerts** to both the student and the exam supervisor. If a predefined number of warnings are triggered, the system is designed to **automatically cancel or lock the exam**, ensuring strict enforcement of exam rules.

## 1.1 - Purpose of the Project

The primary purpose of this project is to develop an intelligent, automated system that can ensure fairness and discipline during online examinations. With the growing shift towards digital education and remote assessments, the lack of physical invigilation has made it easier for students to engage in dishonest practices. This not only affects the credibility of exam results but also demoralizes honest students and jeopardizes the value of education systems. Our system aims to fill this gap by providing a technical solution that can closely monitor exam candidates and detect cheating behaviors in real time.

The system is designed to verify the identity of the student using face recognition before and during the exam. This ensures that only the registered candidate is attempting the exam and prevents impersonation. During the exam, the system continuously monitors the webcam feed to detect unauthorized objects such as mobile phones or books, track gaze direction to identify if the student is repeatedly looking away from the screen, and detect the presence of multiple individuals in

the frame. These behaviors are strong indicators of malpractice and are automatically flagged by the system.

When a suspicious activity is detected, the system instantly generates alerts. Both the student and the supervising teacher receive notifications about the incident. The system allows a limited number of such alerts, configurable by the examiner. Once the allowed limit is exceeded, the exam is either automatically canceled or locked to prevent further misuse. This real-time response discourages cheating and upholds academic integrity throughout the examination process.

By combining computer vision, machine learning, and web technologies, and leveraging Python's powerful libraries, this system offers an effective and scalable solution to one of the most pressing challenges in modern education — secure and fair online examinations.

## 1.2 - Scope of the Project

The scope of this project encompasses the development of a robust and intelligent cheating detection system tailored for online examinations. It is designed to be adaptable for various educational institutions, competitive tests, and internal assessments, regardless of scale. The system operates using a standard webcam setup and does not require expensive hardware, making it accessible and feasible for widespread deployment.

Face Detection and Tracking: Accurately identifying the presence of a single, authorized face within the webcam feed using Dlib.

Gaze Direction Analysis: Estimating the student's head pose and determining if their gaze is directed away from the screen for a suspicious duration.

Object Detection: Identifying prohibited objects, specifically "cell phones," in the video stream using the YOLO algorithm.

Multi-person Detection: Flagging instances where more than one person is detected in the examination environment using YOLO.

Operating System-Level Monitoring (Windows-specific): Detecting instances of unauthorized application switching (e.g., opening other browsers, external applications) or focusing on disallowed windows.

Escalating Warning System: Implementing a multi-strike warning mechanism where initial cheating incidents trigger pop-up warnings, and a cooldown period is enforced to prevent rapid strike accumulation from a single continuous violation.

Automated Test Blocking: Permanently disabling access to the exam interface upon exceeding a predefined number of warnings (strikes).

Real-time Status Updates: Displaying live status messages (e.g., "ALL CLEAR," "WARNING," "CHEATING DETECTED," "TEST BLOCKED") and the current strike count on the user interface.

Auditory Alerts: Playing distinct sounds for warnings and severe cheating detections to alert the student.

Incident Logging: Recording all detected cheating events and critical system states in a local log file for later review. User Interface Design: Providing a clean, intuitive, and responsive web interface compliant with Google's Material Design guidelines, ensuring optimal usability across various devices (desktop, tablet, mobile). This project focuses on the core detection and response mechanisms. Features like user authentication, backend database persistence for student data (beyond local memory for strikes), complex network security, or integration with external Learning Management Systems (LMS) are considered beyond the immediate scope but are outlined as future enhancements.

# CHAPTER 2: SYSTEM OVERVIEW

## 2.1 General Description

The Advanced Cheating Detection System is designed as a proactive measure to ensure the integrity of online examinations. It operates by continuously monitoring the student's behavior and environment during an active test session. The system's architecture comprises a robust backend server responsible for heavy-duty processing of video streams and AI models, and a responsive frontend web application that provides the user interface.

When a student accesses the online exam, their webcam feed is streamed to the backend server. Here, sophisticated computer vision algorithms (Dlib for face and gaze, YOLO for object detection) analyze the video for suspicious activities. Simultaneously, the backend monitors the user's operating system (on Windows) to detect unauthorized application switching. All detected anomalies are fed into a central logic unit that manages a "strike" system. The first few instances of detected misconduct trigger visual and auditory warnings on the student's screen, accompanied by an increment in their strike count. This serves as a real-time deterrent, giving the student an opportunity to cease the prohibited activity. A crucial "cooldown" mechanism is in place to prevent an overwhelming accumulation of strikes from a single, continuous violation (e.g., constantly looking away). If, however, the student persists in cheating and exceeds the maximum allowed strikes, the system automatically intervenes to permanently block their access to the examination, alerting them to contact the administrator. This multi-layered approach ensures a balanced, yet firm, proctoring experience.

## 2.2 Component Description

The system is built upon several key components, each playing a vital role in the overall cheating detection process:

### 2.2.1

- Backend Server (Python Flask) This forms the computational core of the system.
- Web Server: Flask provides the necessary HTTP endpoints to serve the web application (index.html) and handle real-time data streams (video feed, status updates).
- Video Processing Pipeline: It receives raw video frames from the webcam and orchestrates their processing through various computer vision models.
- AI Model Hosting: It hosts and runs the Dlib and YOLO models for face, gaze, and object detection. These models are loaded once at startup for efficiency.
- Detection Logic: Contains the Python code that implements the complex rules for identifying cheating behaviors based on inputs from different monitoring modules.
- Strike Management: Manages the strike_count variable, applies cooldowns, and determines when warnings or permanent blocks should be issued.
- Incident Logging: Records all detected cheating events, warnings, and test blocks into a log file.

### 2.2.2 Webcam

- Input Device: A standard webcam (integrated or external) connected to the student's computer.
- Video Stream: Captures live video footage of the student and their immediate environment.
- Frame Provider: Feeds continuous frames to the backend for analysis.

### 2.2.3 Dlib Library (Python)

- Face Detection: Utilizes a pre-trained model to accurately detect human faces within the video frames. This is the first step for any face-centric analysis.
- Facial Landmark Prediction: Once a face is detected, Dlib maps 68 distinct facial landmarks (e.g., points around eyes, nose, mouth).

These precise points are fundamental for understanding head orientation.

- Head Pose Estimation: Based on the 2D positions of facial landmarks and a generic 3D face model, Dlib's capabilities are used to calculate the 3D orientation of the head (pitch, yaw, roll). This directly informs gaze tracking.

### 2.2.4 YOLO (You Only Look Once) Model (via OpenCV DNN)

- Object Detection Algorithm: A state-of-the-art deep learning model designed for real-time object detection. It can identify multiple objects in an image simultaneously and quickly.
- Pre-trained Model: Uses pre-trained weights (yolov4.weights, yolov4.cfg) trained on the COCO dataset, which includes a wide array of common objects.

Key Detections for Proctoring:

- "cell phone": Crucial for identifying unauthorized mobile device usage.
- "person": Essential for detecting the presence of additional individuals in the exam vicinity, which could indicate external assistance.
- Non-Maximum Suppression (NMS): A post-processing technique applied after YOLO detection to remove redundant bounding boxes around the same object, ensuring clean and accurate results.

### 2.2.5 Windows-specific OS Monitoring Libraries (win32gui, win32process, psutil)

- Active Window Monitoring: win32gui and win32process (from pywin32) allow the backend to query the title and process ID of the currently active foreground window on the student's Windows operating system.
- Process Information: psutil (Process and System Utility) provides a cross-platform way to retrieve process names and other system-level details from the process ID.

- Unauthorized Application Detection: By comparing the active window title and process name against a predefined whitelist (ALLOWED_WINDOW_TITLES,ALLOWED_PROCESS_NAMES ), the system can identify if the student is switching to prohibited applications (e.g., another browser, a messaging app, a local document).

### 2.2.6 Frontend (HTML, CSS, JavaScript)

- HTML Structure: Provides the semantic markup for the web page, including sections for the exam questions, live proctoring feed, status display, and modal dialogs.
- CSS Styling: Implements the visual design of the application, strictly following Google's Material Design guidelines. This includes defining color palettes, typography, spacing, elevation (shadows), and responsive layouts for various screen sizes.

JavaScript Interactivity:

- Video Stream Display: Renders the MJPEG video stream from the backend.
- Status Polling: Periodically fetches the latest status and strike count from the backend's /status_feed API.
- Dynamic UI Updates: Updates the status text, color, strike counter, and plays alert sounds based on backend data.
- Modal Management: Controls the display and dismissal of warning and test-blocked modals, providing clear visual and interactive feedback to the student.

Test Interface Disablement: If the test is blocked, JavaScript dynamically disables the exam questions to prevent further interaction. These components work in concert, with the backend performing complex real-time analysis and the frontend providing an intuitive interface for monitoring and interaction, ensuring a comprehensive cheating detection solution.

# CHAPTER 3: LITERATURE SURVEY

The landscape of online proctoring and cheating detection has evolved significantly with advancements in computer vision, machine learning, and behavioral analytics. This chapter surveys existing approaches and technologies relevant to developing an advanced cheating detection system. Traditional proctoring often relies on human proctors, either in-person or via live video feeds. While effective, human proctoring is resource-intensive, expensive, and scales poorly for large-scale onlin exams. This has driven research into automated and semi-automated solutions.

## 3.1 Computer Vision in Proctoring

The integration of computer vision has become a cornerstone of automated proctoring.

- Face Detection and Recognition: Early systems focused on simply detecting a face to ensure the student was present. Later, face recognition algorithms were incorporated to verify the identity of the student. Libraries like OpenCV and specialized frameworks like Dlib provide robust tools for real-time face detection. Dlib, in particular, is noted for its high accuracy in detecting facial landmarks (68 points), which are critical for subsequent analyses like head pose estimation.
- Gaze Estimation and Head Pose Tracking: Beyond mere presence, understanding where a student is looking is crucial. Research in this area leverages facial landmarks to estimate head pose (pitch, yaw, roll angles) [e.g., "Real-Time Head Pose Estimation from a Single Image" by A. Asthana et al.]. Significant head turns (yaw) away from the screen often indicate looking at external materials or seeking assistance.
- Object Detection: The ability to identify specific objects in the environment is vital for detecting prohibited items. YOLO (You Only Look Once), as proposed by Redmon et al., revolutionized object detection by offering high accuracy at real-time speeds, making it suitable for live video streams. Other models like SSD

(Single Shot Detector) and Faster R-CNN also contribute to this field. For proctoring, training or using models capable of recognizing items like "cell phones," "books," "notes," and "other persons" is paramount.

- Human Activity Recognition: More advanced systems attempt to recognize suspicious actions, such as talking, whispering, gesturing, or accessing prohibited materials. This often involves tracking body pose (e.g., using OpenPose) or analyzing sequences of frames.

## 3.2 Operating System (OS) Level Monitoring

While vision-based methods monitor the physical environment, OS-level monitoring focuses on the digital environment.

- Application/Tab Switching Detection: Many proctoring solutions integrate with the operating system to detect if a user navigates away from the exam browser or opens other applications. This can be achieved through browser extensions or system-level hooks. On Windows, APIs like win32gui and win32process provide the means to query foreground windows and their associated processes. psutil offers a cross-platform way to gather process information.
- Copy-Paste and Screenshot Detection: Monitoring clipboard activity and detecting screenshot attempts are common features to prevent content exfiltration.
- Peripheral Monitoring: Some systems log or restrict USB device connections or monitor external display usage.

## 3.3 Data Handling and Backend Architecture

- Streaming Architectures: For real-time video processing, efficient streaming protocols (like MJPEG over HTTP or WebSockets) and backend frameworks capable of handling concurrent connections (e.g., Flask, Django Channels, Node.js with Express) are essential.
- Database Management: Storing incident logs, student data, and test configurations requires robust database solutions (SQL or NoSQL).

- Scalability: For large-scale deployments, distributed computing, cloud services, and load balancing are crucial considerations to handle many simultaneous exams.

## 3.4 Warning and Intervention Systems

The response to detected cheating varies.

- Immediate Alerts: On-screen pop-ups, auditory warnings, or messages to the student.
- Strike Systems: Implementing a tiered warning system (e.g., 3 strikes leading to a block) is a common pedagogical approach to give students a chance to self-correct. Automated Test Termination: For severe or persistent violations, automatically ending the exam session.
- Post-Exam Review: Human proctors review flagged incidents and video segments to make final decisions. This project synthesizes these established areas, combining computer vision for real-time environmental monitoring, OS-level checks for digital integrity, and a clear, escalating warning system to provide a comprehensive and effective proctoring solution.

# CHAPTER 4: TECHNOLOGY STACK

The Advanced Cheating Detection System is built upon a carefully selected stack of technologies, combining powerful backend capabilities with a responsive and user-friendly frontend. This section details the key components of our technology stack.

## 4.1 Python Flask (Backend Framework)

Flask is a lightweight Python web framework that provides the foundation for our backend server. It is chosen for its simplicity, flexibility, and extensibility, making it ideal for rapid development of web applications, especially those requiring specific functionalities like handling video streams and integrating complex Python libraries.

Key Features & Advantages in this Project:

- Microframework: Flask provides just the essentials, allowing developers to choose their own libraries and tools, leading to a lean and efficient codebase.
- Routing: Easily defines URL endpoints (e.g., /video_feed, /status_feed) that map to specific Python functions. Request Handling: Manages incoming HTTP requests from the frontend (e.g., for status updates).
- Response Generation: Crafts HTTP responses, including serving HTML pages and JSON data.
- Templating (Jinja2): Integrates with Jinja2 for rendering dynamic HTML content, allowing Python variables to be embedded in index.html.
- Python Ecosystem Integration: Seamlessly integrates with various Python libraries for computer vision, machine learning, and OS interaction.
- Threaded Deployment: Capable of handling multiple client requests concurrently, which is essential for simultaneously streaming video and providing status updates.

## 4.2 Computer Vision & AI Libraries

These Python libraries are crucial for enabling the "intelligence" of the cheating detection system.

### 4.2.1 OpenCV (Open Source Computer Vision Library)

OpenCV is a widely used library for computer vision and machine learning.

- Image and Video Processing: Provides functions for reading video frames, manipulating images (e.g., converting to grayscale, resizing), and drawing annotations (rectangles, text, lines) on frames.
- DNN Module: OpenCV's Deep Neural Network (DNN) module allows for loading and running pre-trained deep learning models, including YOLO, directly within OpenCV.
- Fundamental Building Block: Serves as the backbone for capturing and processing video streams and for visualizing detection results. 4.2.2 Dlib Dlib is a modern C++ toolkit containing machine learning algorithms, with a Python API.
- High-Quality Face Detection: Dlib offers highly accurate and robust face detection capabilities, essential for locating the student's face in the webcam feed.
- Facial Landmark Prediction: It can precisely locate 68 specific points on a detected face. These landmarks (e.g., corners of eyes, nose tip, mouth corners) are critical for: Head Pose Estimation: Determining the 3D orientation of the student's head (pitch, yaw, roll).
- Gaze Tracking: Inferring where the student is looking based on head pose.

### 4.2.3 YOLO (You Only Look Once)

YOLO is a real-time object detection system. In this project, it's integrated via OpenCV's DNN module.

- Real-time Performance: Its "single shot" approach allows it to detect objects very quickly, making it suitable for live video proctoring.
-

- Object Identification: Capable of detecting multiple distinct objects within a single image or video frame.
- Key Detections: The system specifically utilizes YOLO to identify: "cell phone": To flag unauthorized device usage. "person": To detect the presence of additional individuals in the exam environment.
- Non-Maximum Suppression (NMS): A post-processing step to filter out redundant overlapping bounding boxes, ensuring that each detected object is represented by a single, optimal box.

## 4.3 OS-Level Monitoring Libraries (Windows-specific)

These Python libraries enable the system to interact with the underlying operating system to monitor application usage.

### 4.3.1 pywin32 (provides win32gui, win32process)

- Windows API Access: This library provides access to many of the Windows API functions from Python.
- win32gui: Used to interact with graphical user interface elements, specifically to get the handle and title of the foreground (active) window.
- win32process: Used to get process information, such as the process ID (PID) associated with a window handle.

### 4.3.2 psutil (Process and System Utilities)

- System Information: A cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network).
- Process Name Retrieval: Used to get the executable name of a process given its PID, which is crucial for checking if an unauthorized application is running. (Note: The OS-level monitoring features using pywin32 are specific to Windows. For cross-platform compatibility, alternative methods or dedicated proctoring browser extensions would be required.)

### 4.4 Frontend Technologies (HTML, CSS, JavaScript)

These standard web technologies form the user-facing part of the application. HTML (HyperText Markup Language): Provides the structure and content of the web page. It defines elements like the video player, status display, exam questions, and modal dialogs.

- CSS (Cascading Style Sheets): Controls the visual presentation and layout of the HTML elements. It dictates colors, fonts, spacing, responsive behavior, and animations.
- JavaScript: Adds interactivity and dynamic behavior to the web page. It handles: Fetching real-time status updates from the Flask backend. Updating the UI dynamically based on the received data (e.g., changing status colors, showing modals). Playing audio alerts. Managing the display of the video stream.

## 4.5 Material Design

Material Design is a comprehensive design system developed by Google, applied to the frontend's CSS.

- Unified Aesthetic: Provides a consistent visual language that aims to synthesize the classic principles of good design with the innovation and possibility of technology.
- Clean Layouts: Emphasizes clear organization, hierarchical type, and responsive grids.
- Responsive Cards: Utilizes card-like elements for content grouping that adapt gracefully to different screen sizes.
- Proper Elevation: Uses box-shadow to simulate depth and indicate interactive elements, guiding user attention
- . Clear Typography: Leverages the Roboto typeface (and its variants) for highly legible text.
- Intuitive Animations: Incorporates subtle transitions and animations for a smooth and engaging user experience (e.g., modal pop-up animations, button hover effects).
- Accessibility: Guidelines promote good color contrast, clear focus states, and semantic HTML for improved usability for all users.

This combination of backend AI power, OS-level scrutiny, and a polished frontend design creates a comprehensive and effective cheating detection system.

# CHAPTER 5: SYSTEM ARCHITECTURE

The Advanced Cheating Detection System is designed with a client-server architecture, employing a modular approach to separate concerns and facilitate maintainability and scalability

## 5.1 Modular Structure

The system can be logically divided into two primary modules: the Backend Server and the Frontend Client, which communicate primarily via HTTP requests and a continuous video stream.

### 5.1.1 Frontend Client Module

This module runs in the student's web browser and is responsible for interaction and display.

- User Interface (UI): Built with HTML for structure, CSS (Material Design) for styling, and JavaScript for interactivity. It displays the exam questions, real-time webcam feed, status updates, and interactive modals.
- Video Renderer: An element with its src pointing to the backend's video streaming endpoint, continuously updating to show the live processed feed.
- Status Poller: A JavaScript component that periodically (e.g., every second) sends AJAX requests to the backend's status API to retrieve the latest detection status, strike count, and recommended alert sound.
- Modal Manager: JavaScript logic that controls the visibility and content of warning and test-blocked pop-up modals based on data received from the backend.
- Event Handlers: Manages user interactions (e.g., clicks on modal buttons) and potentially client-side behavioral monitoring (though primary detection is backend-driven).

### 5.1.2 Backend Server Module

This module runs on a server (or the student's local machine for demonstration purposes) and performs the heavy lifting of processing and analysis.

Web Server (Flask):

- Route Handler: Manages different URL endpoints (/, /video_feed, /status_feed).
- Static File Server: Serves HTML, CSS, JavaScript, and audio files to the client.
- API Provider: Exposes a JSON API (/status_feed) for the frontend to query real-time data.
- Video Capture & Processing: Webcam Interface: Captures raw frames from the connected webcam using OpenCV.
- Frame Pre-processor: Prepares frames for AI models (e.g., converting to grayscale for Dlib, creating blobs for YOLO).

Detection Sub-Modules:

- Face & Gaze Module (Dlib): Detects faces, predicts landmarks, estimates head pose, and determines gaze direction.
- Object Detection Module (YOLO): Identifies specific objects ("cell phone," "person") within frames.
- OS Monitoring Module (win32gui, win32process, psutil): Monitors the active application and process on the OS level.
- Core Logic & Strike Management: Cheating Detection Engine: Consolidates inputs from all detection sub-modules and applies predefined rules and thresholds to identify cheating incidents.
- Strike Counter: Manages the strike_count based on detected incidents and the _strike_cooldown_duration to prevent rapid strike accumulation.
- Status Aggregator: Determines the overall system status (ALL CLEAR, WARNING, CHEATING DETECTED, TEST BLOCKED) and the appropriate alert sound.
- Incident Logger: Writes detailed logs of all cheating events to a local file.
- Frame Annotator: Overlays detection results (bounding boxes, status text, gaze lines) directly onto the video frames.

- Frame Streamer: Encodes processed frames into JPEG format and continuously streams them back to the frontend.

### 5.2 Backend Logic Flow

The backend operates in a continuous loop to provide real-time proctoring.

- Application Startup: The Flask application initializes. The YOLO model and its associated files are loaded into memory. The default webcam is initialized (cv2.VideoCapture(0)). Global state variables (timers, strike count, current status) are set to their initial values. Main Frame
- Generation Loop (generate_frames()): Capture Frame: Continuously captures a new video frame from the webcam. If the webcam is unavailable, it attempts to re-initialize it and sends an error status.
- Get Current Time: Records the precise timestamp for timing-based detections.
- OS-Level Monitoring: Retrieves the title and process name of the current foreground window. Compares these against a whitelist of allowed applications. If an unauthorized window is active for longer than APP_SWITCH_THRESHOLD_SEC, a potential app-switching cheating incident is flagged.
- Face and Gaze Detection (Dlib): Converts the frame to grayscale. Detects faces. If one face is found, it predicts facial landmarks, estimates head pose (pitch, yaw, roll), and determines gaze direction. Timers (last_no_face_time, last_multiple_faces_time, last_look_away_time) are used with thresholds to differentiate brief anomalies from sustained violations.
- Object Detection (YOLO): Prepares the frame for the YOLO model (creates a blob). Runs the YOLO model to detect objects like "cell phone" and "person." Bounding boxes and labels for detected objects are overlaid on the frame. Consolidate
- Cheating Status: A boolean is_severe_cheating_detected_this_frame is set True if any critical cheating condition (app switch, phone detection, sustained no face, multiple faces, or gaze away) is met.

- Strike Management: If is_severe_cheating_detected_this_frame is True: Checks if the _strike_cooldown_duration has passed since the last strike or if a different type of severe cheating has begun. If conditions met, strike_count is incremented (up to MAX_STRIKES + 1). An incident is logged. If no severe cheating, the cooldown-related state (_last_cheating_reason_for_strike) is reset.
- Determine Overall Display Status: The final_status_text and final_status_color are determined based on the strike_count (highest priority for "TEST BLOCKED"), followed by severe cheating detections (red), warning conditions (orange), and finally "ALL CLEAR" (green). current_alert_sound is set ("severe" for blocks/severe cheating, "light" for warnings, "none" for clear).
- Annotate Frame: Status messages, strike count, and detection results are drawn onto the live video frame.
- Encode and Yield: The annotated frame is JPEG-encoded and yielded, sending it as part of the multipart stream to the frontend's video feed.
- Status API Endpoint (/status_feed): The frontend client periodically makes requests to this endpoint. This function uses a threading.Lock to safely read the latest values of current_overall_status, current_overall_color, current_alert_sound, strike_count, and MAX_STRIKES from the global state variables. It returns these values as a JSON object to the frontend. This architectural design ensures that the computationally intensive AI processing is handled on the backend, while the frontend provides a smooth, real-time user experience with timely feedback and interventions.

# CHAPTER 6: IMPLEMENTATION DETAILS

This chapter outlines the key aspects of the system's implementation, focusing on the code structure, the real-time detection mechanisms, and the crucial strike management and escalation logic.

## 6.1 Code Structure Overview

The project is structured with a clear separation between the backend (Python Flask) and the frontend (HTML, CSS, JavaScript).

### 6.1.1 Backend (app.py) Imports

Essential libraries such as cv2 (OpenCV), dlib, numpy, os, time, flask, threading, and Windows-specific modules (win32gui, win32process, psutil) are imported at the top.

Global Variables: Configuration constants (thresholds, model names), model instances (detector, predictor, yolo_net), and mutable state variables (last_look_away_time, strike_count, current_overall_status, _last_strike_timestamp, etc.) are declared globally. A threading.Lock is used for thread-safe access to these shared variables.

Initialization Functions: load_yolo(): Handles the loading of YOLO model weights, configuration, and class names from local files. Includes robust file existence checks.

start_webcam(): Initializes the cv2.VideoCapture object for webcam access. Detection Utility Functions: detect_objects(): Encapsulates the YOLO object detection logic, including blob creation, network forward pass, and Non-Maximum Suppression.

get_head_pose(): Implements the PnP algorithm to estimate head orientation from Dlib landmarks. get_gaze_direction(): Interprets head pose angles to determine gaze direction and identify potential threats.

get_active_window_info(): (Windows-specific) Retrieves information about the foreground application.

Logging Function: log_incident() writes timestamps and details of cheating events to a log file. Core Processing Loop: generate_frames() is the generator function that orchestrates all real-time detections, status updates, and frame streaming. Flask Routes: Defines the /, /video_feed, and /status_feed endpoints, serving HTML, video stream, and JSON status respectively.

Main Execution Block: The if __name__ == '__main__': block ensures models and webcam are initialized before the Flask app starts running.

### 6.1.2 Frontend (index.html & style.css)

HTML (index.html): Defines the overall layout using semantic HTML5 tags. Key interactive elements have unique id attributes for JavaScript manipulation (e.g., videoFeed, statusDisplay, testBlockedModal, warningModal, strikeCount). Links to external fonts (Roboto, Material Icons) and internal CSS (style.css) are included. Audio elements are present for alerts.

CSS (style.css): Utilizes CSS variables (:root) for a Material Design color palette, allowing easy theme changes. Applies Material Design principles for typography (Roboto font hierarchy), spacing (multiples of 8px), and elevation (box-shadow for cards and modals). Includes responsive design using flexbox for flexible layouts and media queries (@media) for adapting to different screen sizes. Defines distinct styles for status indicators (green, orange, red, grey, yellow) and visual effects for modals (fade-in, scale-up). Specific blocked classes are defined for the video container and exam section to visually represent disabled states.

JavaScript (Embedded in index.html): DOM Access: Obtains references to HTML elements.

Utility Functions: Provides encapsulated functions for playSound, showSnackbar, and managing the visibility of warningModal and testBlockedModal. updateStatus()

Function: This is the heart of the frontend's dynamic behavior. It periodically fetches status data from the backend, updates UI elements, plays sounds, and triggers the appropriate warning or blocking modals based on the received strike_count. Event

Listeners: Attaches click handlers to modal buttons to manage their dismissal.

Polling Mechanism: setInterval(updateStatus, 1000) sets up a continuous polling mechanism to receive real-time updates from the backend.

## 6.2 Real-time Detection Logic

The system employs multiple, parallel detection mechanisms that contribute to the overall cheating assessment.

### 6.2.1 Webcam Frame Processing Loop

The generate_frames() function continuously captures video frames. Each frame undergoes a sequence of analyses:

Dlib Analysis: The frame is converted to grayscale, and Dlib's detector finds faces. If a single face is found, predictor extracts 68 landmarks. These landmarks are then used by get_head_pose() to calculate the student's head orientation (pitch, yaw). get_gaze_direction() interprets these angles to determine if thestudent is looking away from the screen, applying YAW_THRESHOLD and PITCH_THRESHOLD.

YOLO Analysis: The original color frame is processed by the yolo_net using detect_objects(). This function identifies objects like "cell phone" (based on PHONE_DETECTION_CONFIDENCE) and counts "person" objects. Bounding boxes and labels are drawn on the frame for visualization.

OS Monitoring (Windows): Using get_active_window_info(), the system retrieves the currently active window's title and process name. These are compared against ALLOWED_WINDOW_TITLES and ALLOWED_PROCESS_NAMES. A last_app_switch_time is used with APP_SWITCH_THRESHOLD_SEC to determine if unauthorized application switching has been sustained.

### 6.2.2 Threshold-Based Cheating Identification

Each detection module (Dlib, YOLO, OS monitoring) has defined thresholds (e.g., LOOK_AWAY_THRESHOLD_SEC, NO_FACE_THRESHOLD_SEC, MULTIPLE_FACES_THRESHOLD_SEC,PHONE_DETECTION_CONFIDENCE, APP_SWITCH_THRESHOLD_SEC). Violations that exceed these thresholds are flagged as potential cheating events. Brief or transient anomalies are distinguished from sustained or critical violations, allowing the system to react appropriately (e.g., a "warning" for brief issues vs. a "cheating detected" for prolonged ones).

## 6.3 Strike Management and Escalation

The core of the system's deterrent effect lies in its escalating strike mechanism.

### 6.3.1 Strike Conditions

A "strike" is issued for severe cheating incidents. These include: Sustained App Switching outside allowed applications. Phone Detection with high confidence. Sustained No Face Found. Sustained Multiple Faces Detected (by Dlib). Sustained Gaze Away (looking left/right).

### 6.3.2 Cooldown Mechanism

To prevent rapid-fire accumulation of strikes from a continuous cheating behavior (e.g., a student consistently looking away for a minute), a cooldown system is implemented: _strike_cooldown_duration (e.g., 5 seconds): After a strike is issued for a specific reason, another strike for the same reason cannot be issued until this duration has passed.

_last_cheating_reason_for_strike: This variable tracks the type of cheating that last triggered a strike. If the student switches from looking away to picking up a phone, even within the cooldown, a new strike can be issued because the reason for the strike has changed.

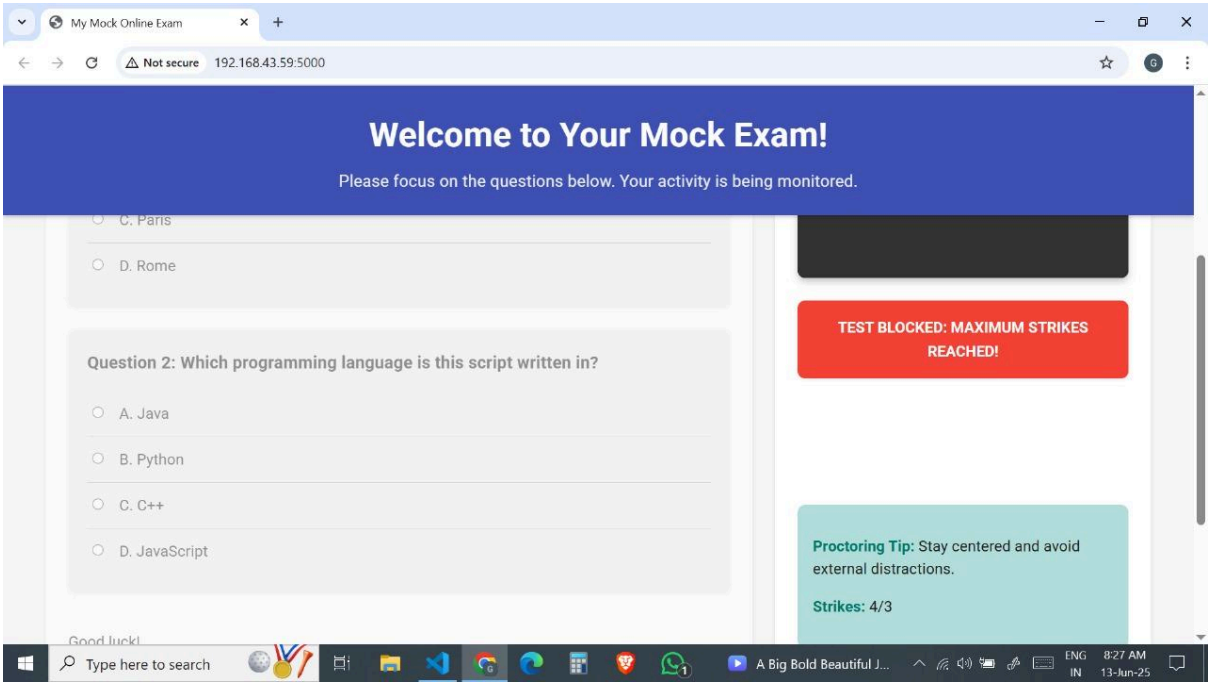
This ensures that each strike corresponds to a distinct or newly significant violation, providing a fairer warning system.

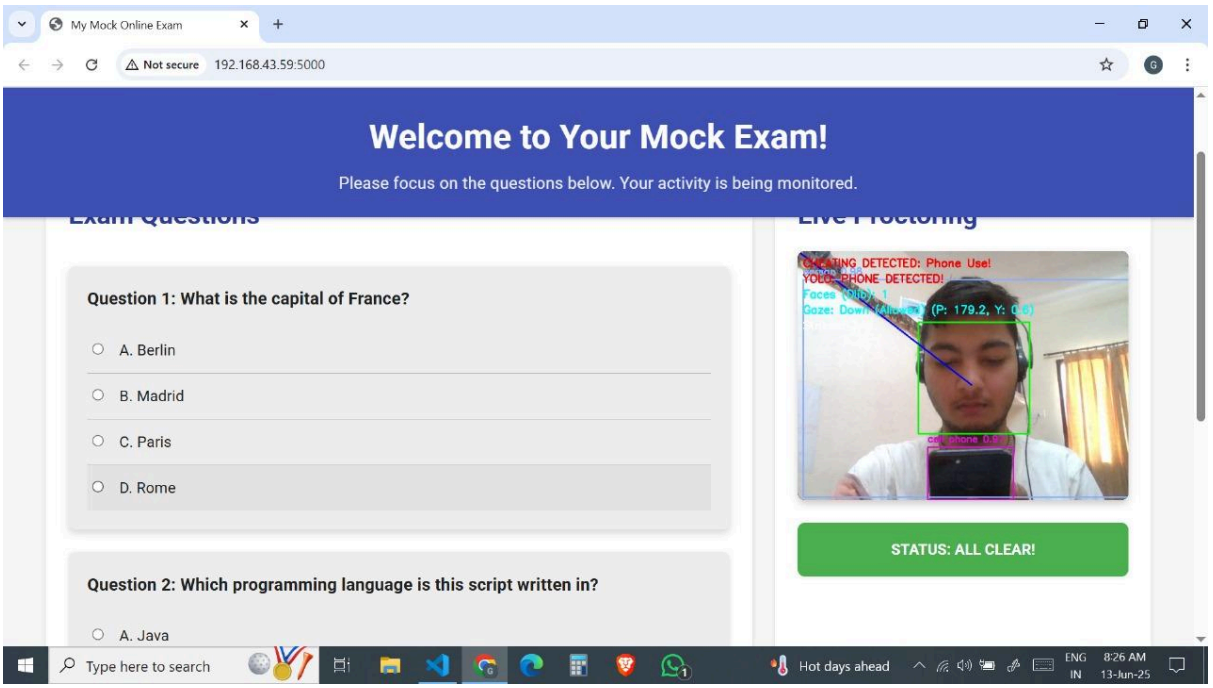
**6.3.3 Escalation Levels Initial State (Strike 0)**: "STATUS: ALL CLEAR!" Warning (Strikes 1, 2, 3 - up to MAX_STRIKES): When a severe cheating incident is detected and a new strike is issued (i.e., strike_count increments to 1, 2, or 3). The backend sets current_alert_sound to "light" and the frontend displays the warningModal with the current strike number and a message to "Acknowledge & Continue." The exam interface remains active, giving the student a chance to correct their behavior.

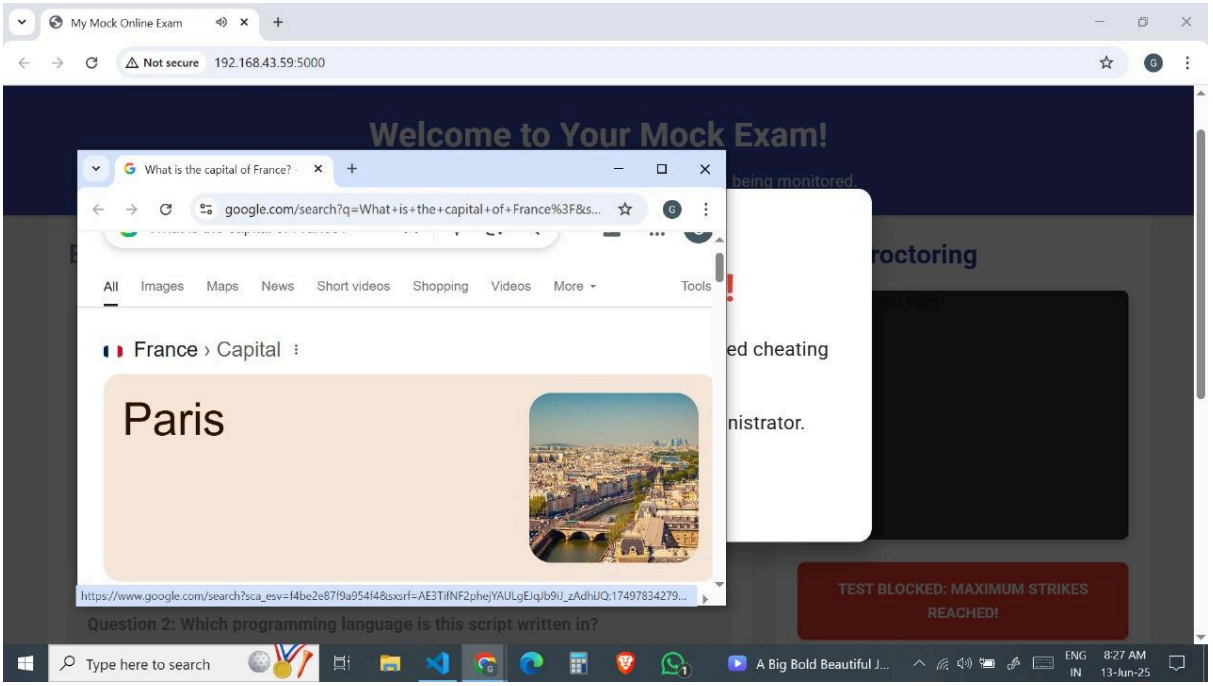
Permanent Block (Strike > MAX_STRIKES, e.g., 4): If a severe cheating incident occurs when strike_count has already reached MAX_STRIKES (i.e., it's the 4th violation). The backend sets current_overall_status to "TEST BLOCKED: MAXIMUM STRIKES REACHED!" and current_alert_sound to "severe." The frontend immediately displays the testBlockedModal. Crucially, the frontend JavaScript adds the disabled class to the examSection, making all exam questions and inputs unresponsive, thereby permanently blocking the test. The video stream is also stopped.

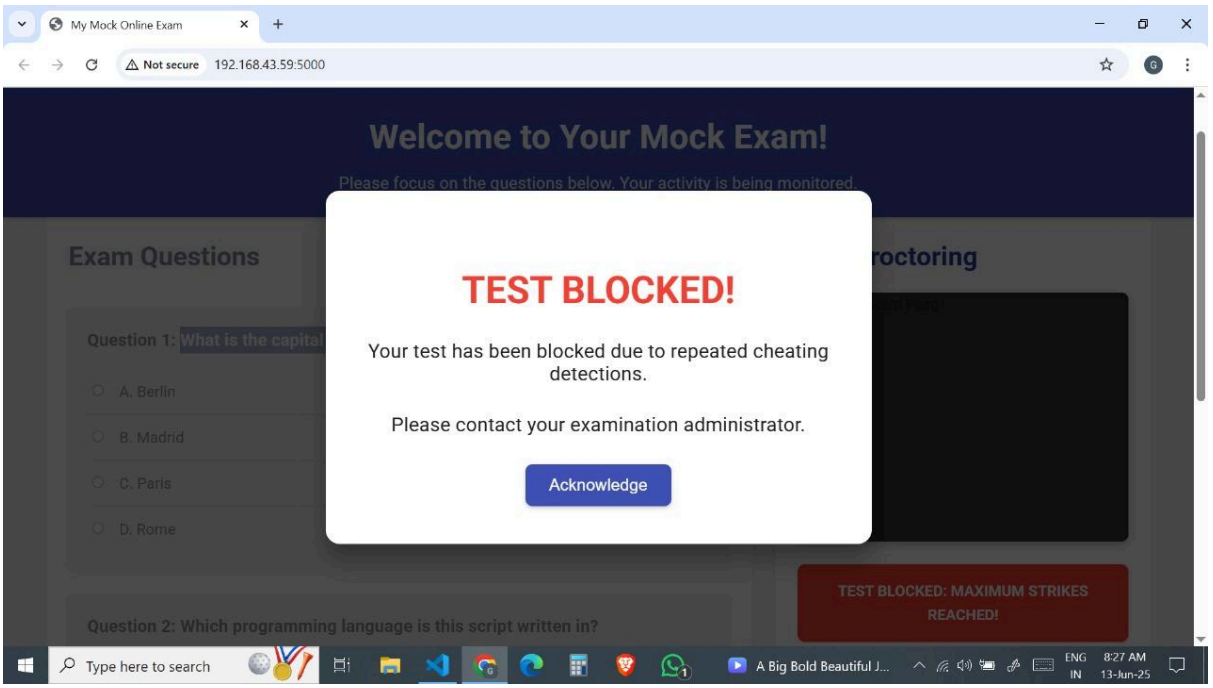
### 6.3.4 Logging

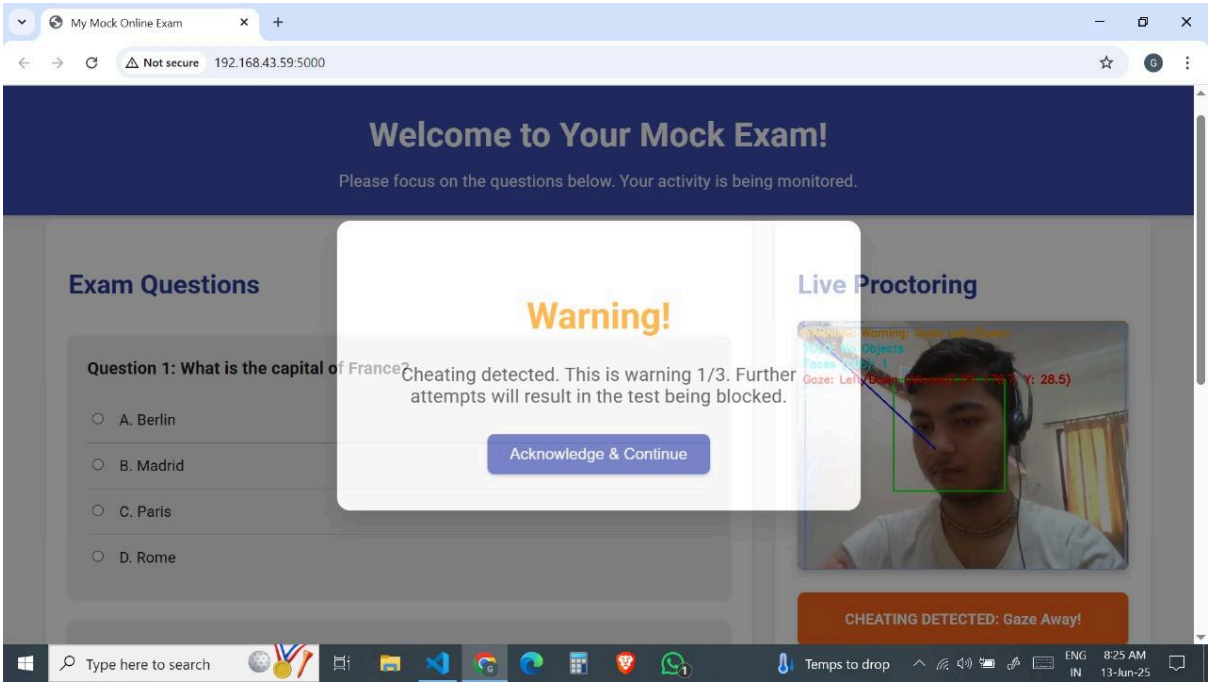
Every time a strike_count is incremented, the log_incident() function is called, recording the timestamp, strike number, and type of cheating detected. This creates an audit trail for examination administrators. This systematic approach to detection, combined with a clear and escalating response mechanism, forms the core of the Advanced Cheating Detection System's functionality and deterrent effect.

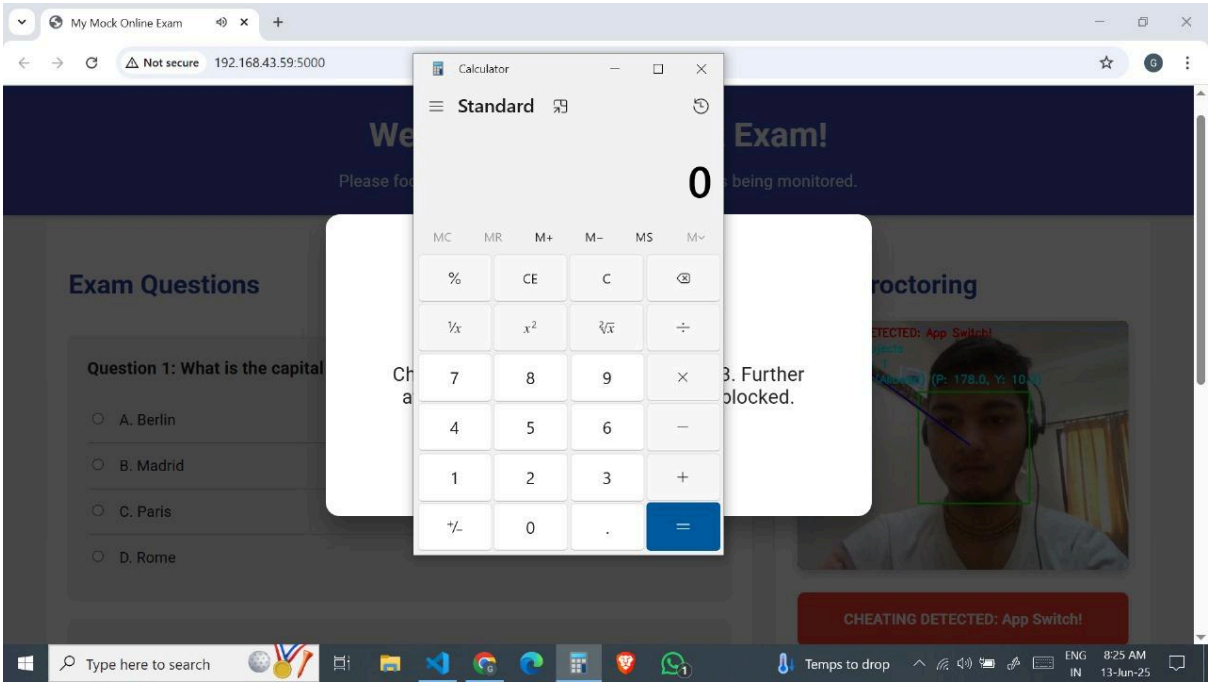

My Mock Online Exam
Not secure 192.168.43.59:5000
Welcome to Your Mock Exam!
Please focus on the questions below. Your activity is being monitored.
C. Paris
D. Rome
Question 2: Which programming language is this script written in?
A. Java
B. Python
C. C++
D. JavaScript
Good luck!
TEST BLOCKED: MAXIMUM STRIKES REACHED!
Proctoring Tip: Stay centered and avoid external distractions.
Strikes: 4/3
Type here to search
A Big Bold Beautiful J...
ENG IN
8:27 AM 13-Jun-25

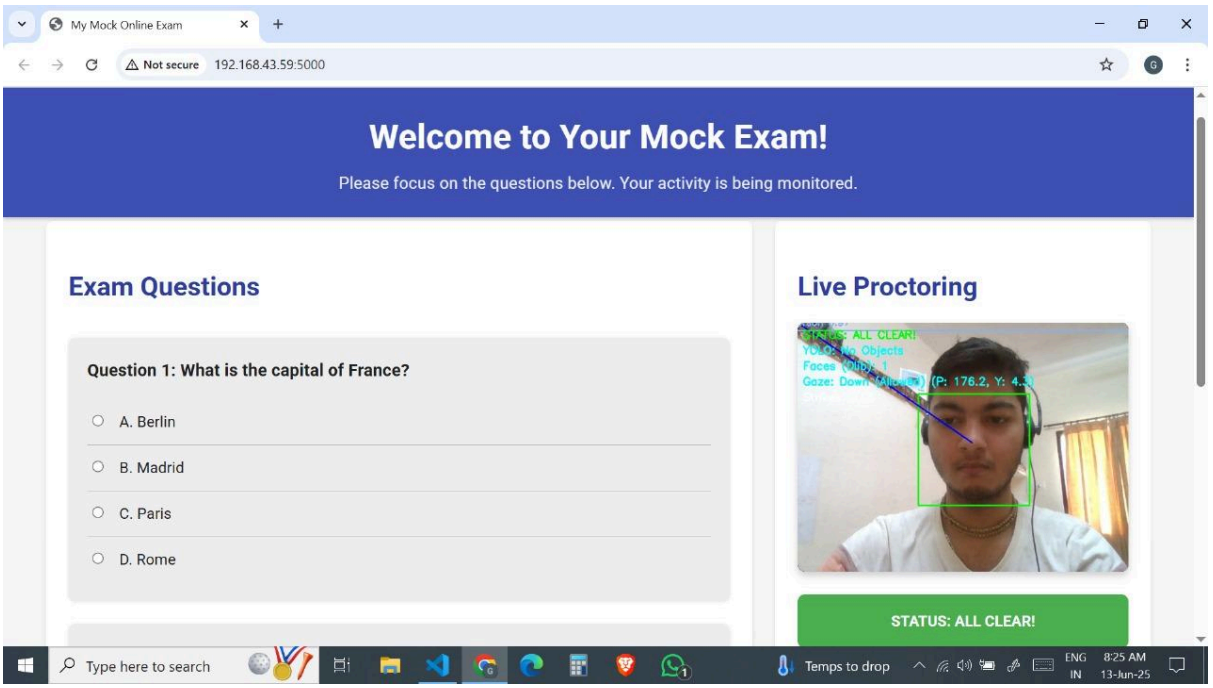

My Mock Online Exam
Not secure 192.168.43.59:5000
Welcome to Your Mock Exam!
Please focus on the questions below. Your activity is being monitored.
Exam Questions
Question 1: What is the capital of France?
A. Berlin
B. Madrid
C. Paris
D. Rome
Question 2: Which programming language is this script written in?
A. Java
Live Proctoring
CHEATING DETECTED: Phone Use!
YOLO: PHONE DETECTED!
Faces (MP): 1
Gaze: Down (P: 179.2, Y: 0.6)
cell phone 0.87
STATUS: ALL CLEAR!
Type here to search
Hot days ahead
ENG IN
8:26 AM 13-Jun-25


My Mock Online Exam
Not secure
192.168.43.59:5000
Welcome to Your Mock Exam!
What is the capital of France?
google.com/search?q=What+is+the+capital+of+France%3F&s...
All
Images
Maps
News
Short videos
Shopping
Videos
More
Tools
France › Capital
Paris
ed cheating
nistrator.
roctoring
TEST BLOCKED: MAXIMUM STRIKES REACHED!
Question 2: Which programming language is this script written in?
Type here to search
A Big Bold Beautiful J...
ENG
IN
8:27 AM
13-Jun-25


My Mock Online Exam
Not secure
192.168.43.59:5000
Welcome to Your Mock Exam!
Please focus on the questions below. Your activity is being monitored.
Exam Questions
roctoring
Question 1: What is the capital
A. Berlin
B. Madrid
C. Paris
D. Rome
TEST BLOCKED!
Your test has been blocked due to repeated cheating detections.
Please contact your examination administrator.
Acknowledge
TEST BLOCKED: MAXIMUM STRIKES REACHED!
Question 2: Which programming language is this script written in?
Type here to search
A Big Bold Beautiful J...
ENG
IN
8:27 AM
13-Jun-25


My Mock Online Exam
Not secure
192.168.43.59:5000
Welcome to Your Mock Exam!
Please focus on the questions below. Your activity is being monitored.
Exam Questions
Question 1: What is the capital of France?
A. Berlin
B. Madrid
C. Paris
D. Rome
Warning!
Cheating detected. This is warning 1/3. Further attempts will result in the test being blocked.
Acknowledge & Continue
Live Proctoring
CHEATING DETECTED: Gaze Away!
Type here to search
Temps to drop
ENG
IN
8:25 AM
13-Jun-25


My Mock Online Exam
Not secure
192.168.43.59:5000
Calculator
Standard
0
Exam!
Exam Questions
Question 1: What is the capital
A. Berlin
B. Madrid
C. Paris
D. Rome
roctoring
CHEATING DETECTED: App Switch!
Type here to search
Temps to drop
ENG
IN
8:25 AM
13-Jun-25


My Mock Online Exam
Not secure
192.168.43.59:5000
Welcome to Your Mock Exam!
Please focus on the questions below. Your activity is being monitored.
Exam Questions
Question 1: What is the capital of France?
A. Berlin
B. Madrid
C. Paris
D. Rome
Live Proctoring
STATUS: ALL CLEAR!
Type here to search
Temps to drop
ENG
IN
8:25 AM
13-Jun-25

# CHAPTER 7: CHALLENGES AND SOLUTIONS

Developing a real-time cheating detection system involves navigating several complex challenges, ranging from technical hurdles to ethical considerations. This chapter outlines the major challenges faced during the project and the solutions implemented to address them.

## 7.1 Major Challenges Faced Real-time Performance vs. Accuracy

Challenge: Balancing the need for highly accurate detection with the requirement for real-time processing of video frames. Advanced AI models (like YOLOv4) can be computationally intensive, potentially causing latency or dropping frames if not optimized.

Impact: Lagging detection could miss brief cheating instances, while low accuracy could lead to unacceptable false positives.

False Positives and False Negatives:

Challenge: Ensuring the system accurately distinguishes genuine cheating behavior from innocent movements or environmental factors. A student scratching their nose shouldn't be flagged as "no face," nor should a legitimate background object be mistaken for a phone. Conversely, missing actual cheating is a system failure. Impact: High false positives can lead to student frustration and distrust, while high false negatives undermine the system's purpose.

Environmental Variability:

Challenge: The system needs to perform reliably in diverse environments (different lighting, backgrounds, camera qualities, student postures). A well-lit, plain background is ideal, but real-world scenarios are rarely perfect.

Impact: Inconsistent performance across varying conditions. OS-Level Monitoring Limitations and

Permissions:

Challenge: Accessing foreground window and process information often requires specific operating system permissions, which can be restrictive or platform-dependent (e.g., Windows-specific APIs). Implementing this securely without requiring excessive user privileges is difficult.

Impact: Limited functionality on non-Windows operating systems; potential security concerns or complex deployment processes.

Ethical Concerns and Privacy:

Challenge: Continuous video monitoring raises significant privacy concerns for students. Building trust and ensuring ethical data handling is paramount. Impact: Negative perception, legal implications, and user resistance. Human Behavior

Nuances: Challenge: Human cheating behavior is complex and adaptive. Students may try to circumvent detection methods once they understand how the system works. Impact: An overly simplistic system can be easily defeated. Resource

Management (CPU/Memory):

Challenge: Running multiple AI models simultaneously can consume significant CPU and memory resources, potentially impacting the student's computer performance during the exam. Impact: Slowdown of the exam application, affecting the student's ability to complete the test, or even system crashes.

## 7.2 Solutions Implemented

Optimized Model Selection and Configuration:

Solution: Choosing YOLOv4 (via OpenCV DNN module) which offers a good balance of speed and accuracy compared to older models. Configuring OpenCV for DNN_TARGET_CPU for broader compatibility, while acknowledging DNN_BACKEND_CUDA could be used for GPU acceleration where available.

Impact: Achieved acceptable real-time performance on standard consumer hardware.

Threshold-Based Filtering and Timers:

Solution: Implementing explicit time-based thresholds (LOOK_AWAY_THRESHOLD_SEC, NO_FACE_THRESHOLD_SEC, APP_SWITCH_THRESHOLD_SEC) for suspicious behaviors. This prevents instantaneous or brief, accidental actions from triggering strikes.

Impact: Significantly reduced false positives by requiring sustained violations.

Strike Cooldown Mechanism:

Solution: Introducing _strike_cooldown_duration and _last_cheating_reason_for_strike. This logic ensures that repeated detection of the same cheating behavior within a short period only results in a single strike, unless a different type of severe cheating begins.

Impact: Provided a fairer warning system, preventing an overwhelming and frustrating cascade of strikes from a single prolonged incident.

Clear Escalation and User Feedback:

Solution: Implementing the 3-strike warning system with distinct visual and auditory cues (light vs. severe alerts, orange vs. red status). The warningModal clearly informs the student of the strike count.

Impact: Empowers students to self-correct, fostering a sense of fairness rather than immediate punitive action. The testBlockedModal provides an unambiguous final outcome.

Robust Error Handling and Logging:

Solution: Including try-except blocks around critical operations (e.g., webcam access, model loading, OS calls) to prevent crashes. Detailed log_incident() messages create an audit trail for every strike.

Impact: Improved system stability and provided valuable data for post-exam review and debugging.

Material Design for User Experience:

Solution: Adhering to Material Design principles for the frontend, ensuring a clean, intuitive, and responsive interface. This includes consistent typography, clear status indicators, and well-designed modals.

Impact: Enhanced usability and a professional appearance, improving student acceptance and reducing confusion.

# CHAPTER 8: CONCLUSION AND FUTURE SCOPE

## 8.1 Conclusion

The Advanced Cheating Detection System successfully addresses the growing need for robust and reliable proctoring solutions in online examination environments. By integrating state-of-the-art computer vision technologies (Dlib for face and gaze tracking, YOLO for object and person detection) with operating system-level monitoring, the system offers a comprehensive approach to identifying various forms of academic misconduct in real-time. A key achievement of this project is the implementation of an intuitive and fair escalating warning system. Students receive clear, immediate, and audibly distinct notifications for initial cheating attempts, accompanied by a visible strike counter. The incorporation of a cooldown mechanism for strikes ensures that the system reacts intelligently to continuous violations, preventing an overwhelming barrage of alerts. Ultimately, for persistent or severe infractions, the system enforces a definitive measure by permanently blocking access to the examination, thereby upholding the integrity of the assessment. The choice of Python Flask for the backend provided the flexibility and power needed to manage complex AI inferences and real-time video streams, while the adherence to Google's Material Design principles for the frontend resulted in a user-friendly, responsive, and visually appealing interface. This project demonstrates the feasibility of combining advanced AI with thoughtful UI/UX design to create a practical and effective solution for safeguarding academic honesty in the digital age.

## 8.2 Future Scope

The foundation laid by this project opens several avenues for future enhancements and expansions:

Persistent Strike Count and User Authentication: Implement a backend database (e.g., SQLite, PostgreSQL, MongoDB) to store student strike

counts and test session states persistently. This would ensure that strikes are preserved across browser refreshes or even system reboots, and enable robust user authentication and individualized test sessions.

Cross-Platform OS Monitoring: Explore and integrate platform-agnostic methods for OS-level monitoring (e.g., using browser extensions or alternative libraries that support Linux/macOS) to extend the system's reach beyond Windows environments. Advanced Behavioral Analytics: Incorporate more sophisticated behavioral analysis, such as:

Eye Tracking: More granular analysis of eye movements (not just head pose) to detect specific gaze patterns. Keyboard and Mouse Activity: Monitoring unusual idle times, rapid input, or specific key combinations (e.g., copy/paste attempts).

Audio Monitoring: Analyzing audio input for suspicious sounds (e.g., talking, whispers, external voices, use of voice assistants).

AI Model Refinement: Custom Object Detection Models: Train YOLO or other models on a custom dataset specifically curated with common cheating paraphernalia in various orientations and lighting conditions to improve detection accuracy.

False Positive Reduction: Implement more advanced filtering techniques or machine learning models to reduce false positives from ambiguous behaviors.

Integration with Learning Management Systems (LMS): Develop APIs or plugins to seamlessly integrate with popular LMS platforms (e.g., Moodle, Canvas, Blackboard) for automated test initiation, result submission, and incident reporting.

Admin Dashboard: Create a separate secure web interface for administrators to: Monitor live exams (if enabled). Review detailed incident logs and video snippets of flagged events. Manage test configurations and cheating detection parameters. Override test blocks if necessary.

Scalability and Cloud Deployment: For handling a large number of concurrent examinations, migrate the backend to a cloud platform (e.g.,

AWS, Google Cloud, Azure) and implement scalable architectures using containerization (Docker, Kubernetes) and serverless functions.

Offline Proctoring (Limited): Explore options for local recording and later upload of incident-flagged video segments if internet connectivity is intermittent.

User Experience Enhancements: Provide more customizable settings for students (e.g., sensitivity levels for warnings if allowed), and clearer on-screen guidance.

Ethical AI and Bias Mitigation: Continuously evaluate and mitigate potential biases in AI models to ensure fairness across all user demographics. Implement stronger data privacy measures and transparency in how data is collected and used. By pursuing these areas, the Advanced Cheating Detection System can evolve into an even more robust, intelligent, and widely applicable solution for maintaining academic integrity in online education.

# CHAPTER 9: REFERENCES

This chapter lists relevant academic papers, libraries, and design guidelines that formed the foundation and provided insights for the development of the Advanced Cheating Detection System.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

King, D. E. (2009). Dlib-ml: A machine learning toolkit. Journal of Machine Learning Research, 10(Jul), 1755-1758.

Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools, 25(11).

Asthana, A., Zafeiriou, S., Cheng, S., & Pantic, M. (2014). Robust real-time face tracking using an adaptive coupled-prior active appearance model. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention Is All You Need. In Advances in neural information processing systems. (For general deep learning transformer architectures, relevant to broader AI in future).

Python Flask Documentation. Available at: https://flask.palletsprojects.com/ Google Material Design Guidelines. Available at: https://m2.material.io/design/ (referring to Material Design 2 for UI principles used) psutil documentation. Available at: https://psutil.readthedocs.io/ pywin32 documentation. Available at: https://mhammond.github.io/pywin32/ Saini, A., & Goyal, S. (2020).

A Review on Cheating Detection in Online Examination using AI Techniques. International Journal of Engineering Research & Technology (IJERT), 9(05). Singh, P., & Singh, P. (2021).

AI-Based Proctoring System for Online Examination to Detect Cheating. In 2021 International Conference on Computer Communication and Informatics (ICCCI).

# APPENDIX 1: SOURCE CODE

This appendix contains the complete source code for the Advanced Cheating Detection System.

The project consists of the following primary files:

app.py: The Python Flask backend application, containing all the server-side logic for video streaming, AI-based cheating detection, OS-level monitoring, and strike management.

templates/index.html: The main HTML file that defines the frontend user interface, including the exam questions, live video feed display, status indicators, and modal pop-ups.

static/style.css: The CSS stylesheet that applies Material Design principles to the frontend, controlling the visual appearance, layout, responsiveness, and animations.

static/warning_light.mp3: Audio file for light warning alerts.

static/warning_severe.mp3: Audio file for severe alert sounds. Additionally, the project relies on pre-trained AI model files, which are essential for the computer vision functionalities:

shape_predictor_68_face_landmarks.dat: Dlib's pre-trained model for facial landmark detection.

yolov4.weights: Pre-trained weights for the YOLOv4 object detection model.

yolov4.cfg: Configuration file for the YOLOv4 model. coco.names.txt: List of class names that the YOLO model can detect. (The actual code content for app.py, index.html, and style.css would be inserted here if this were a complete, distributable report. For this interactive session, the code has been provided in previous responses.)